Advance JAVA Tutorials

www.enosislearning.com

ADVANCE JAVA OVERVIEW

INTRODUCTION TO ADVANCED JAVA

Basically advanced java is divided into five main topics viz Multithreading, Collections, JDBC, Servlet & JSP. Advanced Java is everything that goes beyond Core Java – most importantly the APIs defined in Java Enterprise Edition, includes Servlet programming, Web Services, the Persistence API, etc. It is a Web & Enterprise application development platform which basically follows client & server architecture.

NEED FOR ADVANCE JAVA

Below are the few major advantages of Advance Java:

1. Advance Java i.e. JEE (Java Enterprise Edition) gives you the library to understand the Client-Server architecture for Web Application Development which Core Java doesn't support.
2. J2EE is platform Independent, Java Centric environment for developing, building & deploying Web-based applications online. It also consists of a set of services, APIs, and protocols, which provides the functionality that is necessary for developing multi-tiered, web-based applications.
3. You will be able to work with Web and Application Servers like Apache Tomcat, Glassfish etc and understand the communication over HTTP protocol. But, in Core Java, it is not possible.
4. There are a lot of Advance Java frameworks like Spring, JSF, Struts etc. which enable you to develop a secure transaction based web apps for the domains like E-Commerce, Banking, Legal, Financial, Healthcare, Inventory etc.
5. To work and understand the hot technologies like Hadoop and Cloud services, you should be prepared with core and advanced Java concepts.

CHAPTER 1: MULTITHREADING

INTRODUCTION TO MULTITHREADING

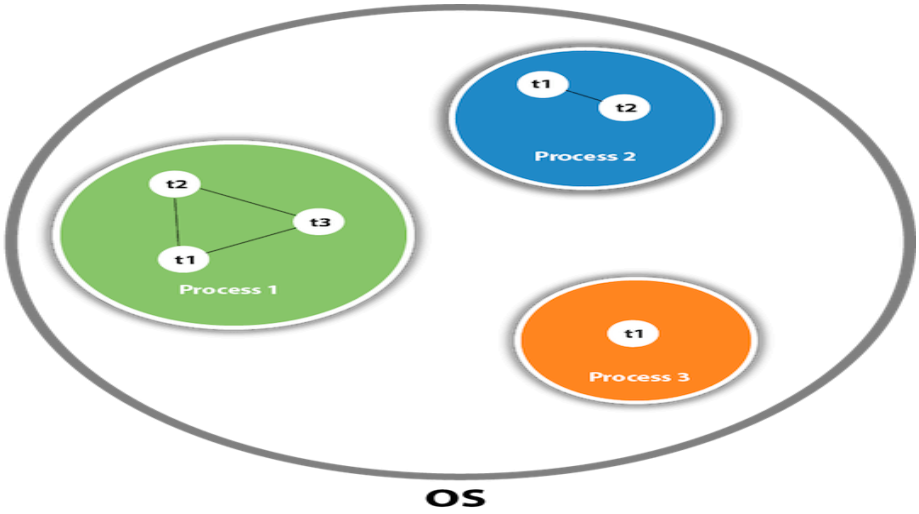
A program can be divided into a number of small processes. Each small process can be addressed as a single thread (a lightweight process). Multithreaded programs contain two or more threads that can run concurrently. This means that a single program can perform two or more tasks simultaneously.

For example, one thread is writing content on a file at the same time another thread is performing spelling check.

WHAT IS THREAD IN JAVA

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

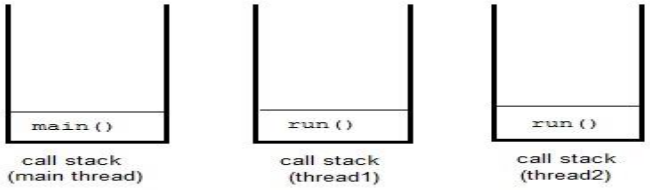


As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

In Java, the word **thread** means two different things.

- An instance of **Thread** class.
- or, A thread of execution.

An instance of **Thread** class is just an object, like any other object in java. But a thread of execution means an individual "lightweight" process that has its own call stack. In java each thread has its own call stack.



The execution of multithreading in stack is as shown in above figure.

THE MAIN THREAD

Even if you don't create any thread in your program, a thread called **main** thread is still created automatically. Although the **main** thread is automatically created, you can control it by obtaining a reference to it by calling **currentThread()** method.

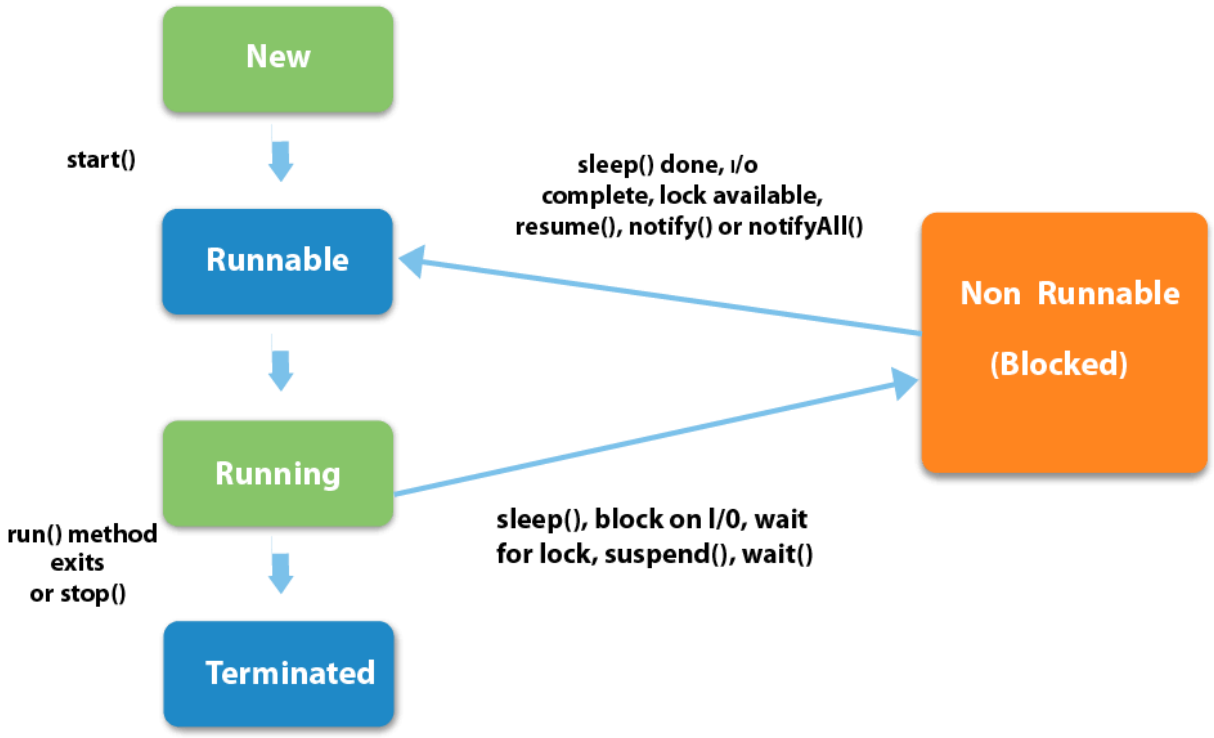
Note : Two important things to know about **main** thread are,

- It is the thread from which other threads will be produced.
- **main** thread must be always the last thread to finish execution.

Example:

```
class MainThread
{
 public static void main(String[] args)
 {
  Thread t=Thread.currentThread();
  t.setName("MainThread");
  System.out.println("Name of thread is "+t);
 }
}
```

LIFE CYCLE OF A THREAD



Above figure gives the propogation of any thread from its generation till termination with different methods.

1. **New:** A thread begins its life cycle in the new state. It remains in this state until the start() method is called on it.
2. **Runable:** After invocation of start () method on new thread, the thread becomes runable.
3. **Running:** A method is in running thread if the thread scheduler has selected it.
4. **Waiting:** A thread is waiting for another thread to perform a task. In this stage the thread is still alive.
5. **Terminated:** A thread enter the terminated state when it complete its task.

THREAD CLASS

Thread class is the main class on which Java's Multithreading system is based. Thread class, along with its companion interface **Runnable** will be used to create and run threads for utilizing Multithreading feature of Java.

Constructors of Thread class

1. **Thread** ( )

2. **Thread** (*String str*)
3. **Thread** (*Runnable r*)
4. **Thread** (*Runnable r*, *String str*)

You can create new thread by any one of the following method:

1. By implementing Runnable interface.
2. By extending Thread class

THREAD METHODS

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
setPriority()	Sets threads priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end
run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

THREAD PRIORITIES

Every thread has a priority that helps the operating system determine the order in which threads are scheduled for execution. In java thread priority ranges between,

- MIN-PRIORITY (a constant of 1)
- MAX-PRIORITY (a constant of 10)

By default every thread is given a NORM-PRIORITY(5). The **main** thread always have NORM-PRIORITY.

We can get and set the priority of any thread with the help of getpriority and setPriority methods of thread.

Important Note:

1. When we extend Thread class, we cannot override **setName ()** and **getName ()** functions, because they are declared final in Thread class.
2. While using **sleep ()**, always handle the exception it throws.

   Static void **sleep** (long *milliseconds*) throws **InterruptedException**

HOW TO CREATE A THREAD

1. IMPLEMENTING THE RUNNABLE INTERFACE

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface , the class needs to implement the **run()** method, which is of form,

public *void* **run**() method:

- run() method introduces a concurrent thread into your program. This thread will end when run() returns.
- You must specify the code for your thread inside run() method.
- run() method can call other methods, can use other classes and declare variables just like any other normal method.

Example:

```
class MyThread implements Runnable
{
 public void run()
 {
  System.out.println("concurrent thread started running..");
 }}


class MyThreadDemo
{
 public static void main( String args[] )
```

```
 {
  MyThread mt = new MyThread();
  Thread t = new Thread(mt);
t.start();
 }
}
```

In above program, To call the **run()** method, **start()** method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program.

2. EXTENDING THREAD CLASS

This is another way to create a thread by a new class that extends **Thread** class and create an instance of that class. The extending class must override **run()** method which is the entry point of new thread.

Example:

```
class MyThread extends Thread
{
 public void run()
 {
  System.out.println("Concurrent thread started running..");
 }
}

classMyThreadDemo
{
 public static void main( String args[] )
 {
  MyThread mt = new  MyThread();
mt.start();
 }
}
```

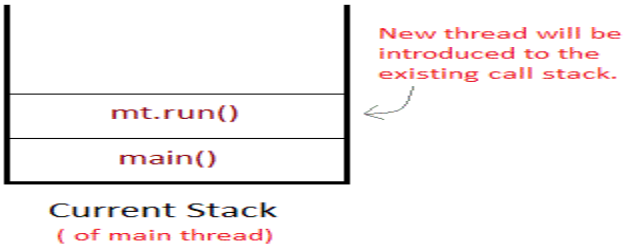
In this case also, as we must override the **run()** and then use the **start()** method to start and run the thread. Also, when you create MyThread class object, Thread class constructor will also be invoked, as it is the **super class**, hence MyThread class object acts as Thread class object.

What if we call run() method directly without using start() method ?

In above program if we directly call **run()** method, without using **start()** method, like this

```
public static void main( String args[] )
{
 MyThread mt = new MyThread();
mt.run();
}
```

Doing so, the thread won't be allocated a new call stack, and it will start running in the current c all stack, that is the call stack of the **main** thread. Hence Multithreading won't be there.



Above figure shows the actual existance of newly generated thread. If we call run method then new thread will not get any stack location.

Can we Start a thread twice ?

No, a thread cannot be started twice. If you try to do so, **IllegalThreadStateException** will be thrown.

Example:

```
public static void main( String args[] )
{
 MyThread mt = new MyThread();
mt.start();
```

```
mt.start();    //Exception thrown
}
```

When a thread is in running state, and you try to start it again, or any method try to invoke that thread again using **start()** method, exception is thrown.

JOINING THREADS

Sometimes one thread needs to know when another thread is ending. In java, **isAlive()** and **join()** are two different methods to check whether a thread has finished its execution.

The **isAlive()** method returns **true** if the thread upon which it is called is still running otherwise it returns **false**.

final boolean **isAlive()**

Example of isAlive method:

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                Thread.sleep(500);
                }
                catch(InterruptedException ie) { }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();
                t2.start();
                System.out.println(t1.isAlive());
```

```
                System.out.println(t2.isAlive());
        }
}
```

But, **join()** method is used more commonly than **isAlive()**. This method waits until the thread on which it is called terminates.

final void **join()** throws **InterruptedException**

Using **join()** method, we tell our thread to wait until the specified thread completes its execution. There are overloaded versions of **join()** method, which allows us to specify time for which you want to wait for the specified thread to terminate.

final void **join**(long *milliseconds*) throws **InterruptedException**

Example of thread without join() method:

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                Thread.sleep(500);
                }catch(InterruptedException ie){ }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();
                t2.start();
        }}
```

In this above program two thread t1 and t2 are created. t1 starts first and after printing "r1" on console thread t1 goes to sleep for 500 ms. At the same time Thread t2 will start its process and print "r1" on console and then go into sleep for 500 ms. Thread t1 will wake up from sleep and print "r2" on console similarly thread t2 will wake up from sleep and print "r2" on console. So you will get output like r1 r1 r2 r2

Example of thread with join() method:

```
public class MyThread extends Thread
{
        public void run()
        {
                System.out.println("r1 ");
                try {
                Thread.sleep(500);
                }catch(InterruptedException ie){ }
                System.out.println("r2 ");
        }
        public static void main(String[] args)
        {
                MyThread t1=new MyThread();
                MyThread t2=new MyThread();
                t1.start();

                try{
                        t1.join(); //Waiting for t1 to finish
                }catch(InterruptedException ie){}

                t2.start();
        }
}
```

In this above program join() method on thread t1 ensures that t1 finishes it process before thread t2 starts.

Specifying time with join()

If in the above program, we specify time while using **join()** with **t1**, then **t1** will execute for that time, and then **t2** will join it.

t1.join(1500);

Doing so, initially t1 will execute for 1.5 seconds, after which t2 will join it.

CONCEPT OF SYNCHRONIZATION

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called **synchronization**. The synchronization keyword in java creates a block of code referred to as critical section.

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

General Syntax :

```
synchronized (object)
{
//statement to be synchronized
}
```

Why we use Syncronization ?

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Scenario without synchronization:-

Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

Solution to above problem:-

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Example with no Synchronization:

```
class First
{
 public void display(String msg)
 {
  System.out.print ("["+msg);
  try
  {
   Thread.sleep(1000);
  }
  catch(InterruptedException e)
  {
   e.printStackTrace();
  }
  System.out.println ("]");
 }
}

class Second extends Thread
{
 String msg;
 First fobj;
 Second (First fp,String str)
 {
  fobj = fp;
  msg = str;
  start();
 }
 public void run()
 {
  fobj.display(msg);
 }
}
```

```
public class Syncro
{
 public static void main (String[] args)
 {
First fnew = new First();
  Second ss = new second(fnew, "welcome");
  Second ss1= new second (fnew,"new");
  Second ss2 = new second(fnew, "programmer");
 }
}
```

In the above program, object **fnew** of class First is shared by all the three running threads(ss, ss 1 and ss2) to call the shared method(*void* **display**). Hence the result is unsynchronized and such situation is called **Race condition**.

Synchronized Keyword

To synchronize above program, we must *serialize* access to the shared **display()** method, making it available to only one thread at a time. This is done by using keyword **synchronized** with display() method.

Synchronized *void* **display** (String msg)

Using Synchronised block

If you have to synchronize access to object of a class that has no synchronized methods, and you cannot modify the code. You can use synchronized block to use it.

```
class First
{
 public void display(String msg)
 {
  System.out.print ("["+msg);
  try
  {
   Thread.sleep(1000);
  }
  catch(InterruptedException e)
  {
```

```
  e.printStackTrace();
 }
 System.out.println ("]");
}
}

class Second extends Thread
{
 String msg;
 First fobj;
 Second (First fp,String str)
 {
  fobj = fp;
  msg = str;
  start();
 }
 public void run()
 {
synchronized(fobj)      //Synchronized block
  {
fobj.display(msg);
  }
 }
}
public class Syncro
{
 public static void main (String[] args)
 {
  First fnew = new First();
  Second ss = new second(fnew, "welcome");
  Second ss1= new second (fnew,"new");
  Second ss2 = new second(fnew, "programmer");
```

```
 }
}
```

Because of synchronized block this program gives the expected output.

INTERTHREAD COMMUNICATION

Java provide benefit of avoiding thread pooling using interthread communication.
The **wait()**, **notify()**, **notifyAll()** of Object class. These method are implemented as **final** in Object. All three method can be called only from within a **synchronized** context.

- **wait()** tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.
- **notify()** wakes up a thread that called wait() on same object.
- **notifyAll()** wakes up all the thread that called wait() on same object.

Difference between wait() and sleep()

wait()	**sleep()**
called from synchronised block	no such requirement
monitor is released	monitor is not released
awake when notify() or notifyAll() method is called.	not awake when notify() or notifyAll() method is called
not a static method	static method
wait() is generaly used on condition	sleep() method is simply used to put your thread on sleep.

Chapter 2: Collection Framework

INTRODUCTION TO COLLECTION FRAMEWORK

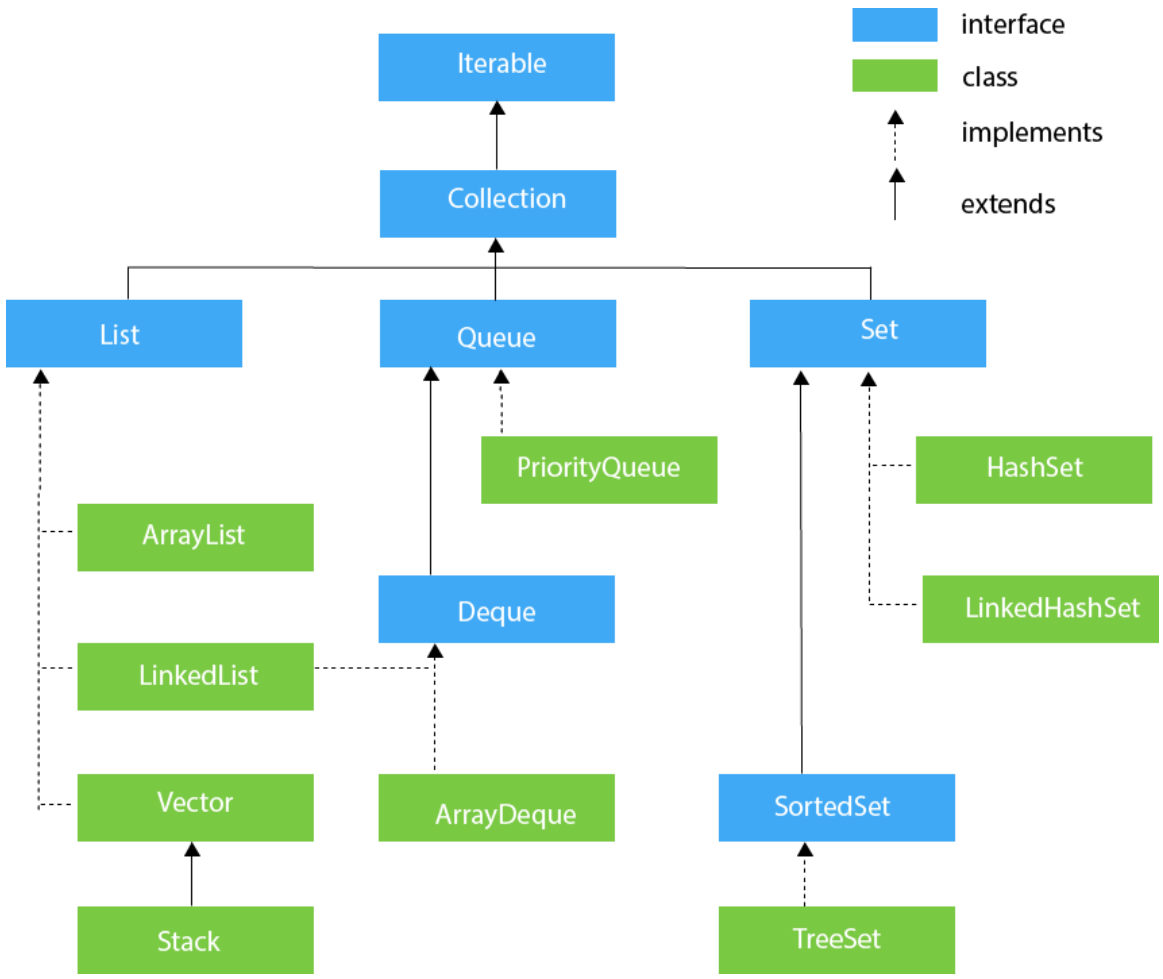
Collections framework was not introduced in earlier Core Java part. A collection was added to J2SE 1.2. Prior to Java 2, Java provided adhoc classes such as Dictionary, Vector, Stack and Properties to store and manipulate groups of objects.

Framework in java is nothing but hierarchy of classes and interfaces. For Collection framework you need to import java.util package. It provides many important classes and interfaces to collect and organize group of alike objects those are introduced in collection framework.

Following table describes the main interfaces or techniques used to store group of objects in collection framework.

Interface	Description
Collection	Enables you to work with groups of object; it is at the top of Collection hierarchy
Deque	Extends Queue to handle double ended queue.
List	Extends Collection to handle sequences list of object.
Queue	Extends Collection to handle special kind of list in which element are removed only from the head.
Set	Extends Collection to handle sets, which must contain unique element.
SortedSet	Extends Set to handle sorted set.

Following figure gives the precise idea about collection framework interfaces & classes.



THE COLLECTION INTERFACE

1. It is at the top of collection heirarchy and must be implemented by any class that defines a collection. Its general declaration is,

   interfaceCollection< E >

2. Following are some of the commonly used methods in this interface.

Methods	Description
boolean add(E obj)	Used to add objects to a collection. Returns true if obj was added to the collection.

	Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.
booleanaddAll(Collection C)	Add all elements of collection C to the invoking collection. Returns true if the element were added. Otherwise, returns false.
boolean remove(Object obj)	To remove an object from collection. Returns true if the element was removed. Otherwise, returns false.
booleanremoveAll(Collection C)	Removes all element of collection C from the invoking collection. Returns true if the collection's elements were removed. Otherwise, returns false.
boolean contains(Object obj)	To determine whether an object is present in collection or not. Returns true if obj is an element of the invoking collection. Otherwise, returns false.
booleanisEmpty()	Returns true if collection is empty, else returns false.
int size()	Returns number of elements present in collection.
void clear()	Removes total number of elements from the collection.
Object[] toArray()	Returns an array which consists of the invoking collection elements.

booleanretainAll(Collection c)	Deletes all the elements of invoking collection except the specified collection.
Iterator iterator( )	Returns an iterator for the invoking collection.
boolean equals(Object obj)	Returns true if the invoking collection and obj are equal. Otherwise, returns false.
Object[] toArray(Object array[])	Returns an array containing only those collection elements whose type matches of the specified array.

THE LIST INTERFACE

Properties of List Interface:

1. It extends the **Collection** Interface, and defines storage as sequence of elements. Following is its general declaration,

   interfaceList< E >

2. Allows random access and insertion, based on position.
3. It allows Duplicate elements.
4. Apart from methods of Collection Interface, it adds following methods of its own.

   Following table shows methods used in List interface.

Methods	Description
Object get(int index)	Returns object stored at the specified index

Object set(int index, E obj)	Stores object at the specified index in the calling collection
intindexOf(Object obj)	Returns index of first occurrence of obj in the collection
intlastIndexOf(Object obj)	Returns index of last occurrence of obj in the collection
List subList(int start, int end)	Returns a list containing elements between start and end index in the collection

THE SET INTERFACE

Properties of Set Interface:

1. This interface defines a Set. It extends **Collection** interface and doesn't allow insertion of duplicate elements. It's general declaration is,

   interfaceSet< E >

2. It doesn't define any method of its own. It has two sub interfaces, **SortedSet** and **NavigableSet**.
3. **SortedSet** interface extends **Set** interface and arranges added elements in an ascending order.
4. **NavigabeSet** interface extends **SortedSet** interface, and allows retrieval of elements based on the closest match to a given value or values.

THE QUEUE INTERFACE

Properties of Queue Interface:

1. It extends **collection** interface and defines behavior of queue, that is first-in, first-out. It's general declaration is,

   interfaceQueue< E >

2. There are couple of new and interesting methods added by this interface. Some of them are mentioned in below table.

Methods	Description
Object pull()	removes element at the head of the queue and returns **null** if queue is empty
Object remove()	removes element at the head of the queue and throws **NoSuchElementException** if queue is empty
Object peek()	returns the element at the head of the queue without removing it. Returns **null** if queue is empty
Object element()	same as peek(), but throws **NoSuchElementException** if queue is empty
boolean offer(E obj)	Adds object to queue.

THE DEQUEUE INTERFACE

1. It extends **Queue** interface and implements behaviour of a double-ended queue. Its general declaration is,

   interfaceDequeue< E >

2. Since it implements Queue interface, it has the same methods as mentioned there.
3. Double ended queues can function as simple queues as well as like standard Stacks.

THE COLLECTION CLASSES

There are some standard classes that implements Collection interface. Some of the classes provide full implementations that can be used as it is. Others are abstract classes, which provide skeletal implementations that can be used as a starting point for creating concrete collections.

The standard collection classes are:

Class	Description
AbstractCollection	Implements most of the Collection interface.
AbstractList	Extends AbstractCollection and implements most of the List interface.
AbstractQueue	Extends AbstractCollection and implements parts of the Queue interface.
AbstractSequentialList	Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.
LinkedList	Implements a linked list by extending AbstractSequentialList
ArrayList	Implements a dynamic array by extending AbstractList
ArrayDeque	Implements a dynamic double-ended queue by extending AbstractCollection and implementing the Deque interface(Added by Java SE 6).
AbstractSet	Extends AbstractCollection and implements most of the Set interface.
EnumSet	Extends AbstractSet for use with enum elements.
HashSet	Extends AbstractSet for use with a hash table.
LinkedHashSet	Extends HashSet to allow insertion-order iterations.
PriorityQueue	Extends AbstractQueue to support a priority-based queue.
TreeSet	Implements a set stored in a tree. Extends AbstractSet.

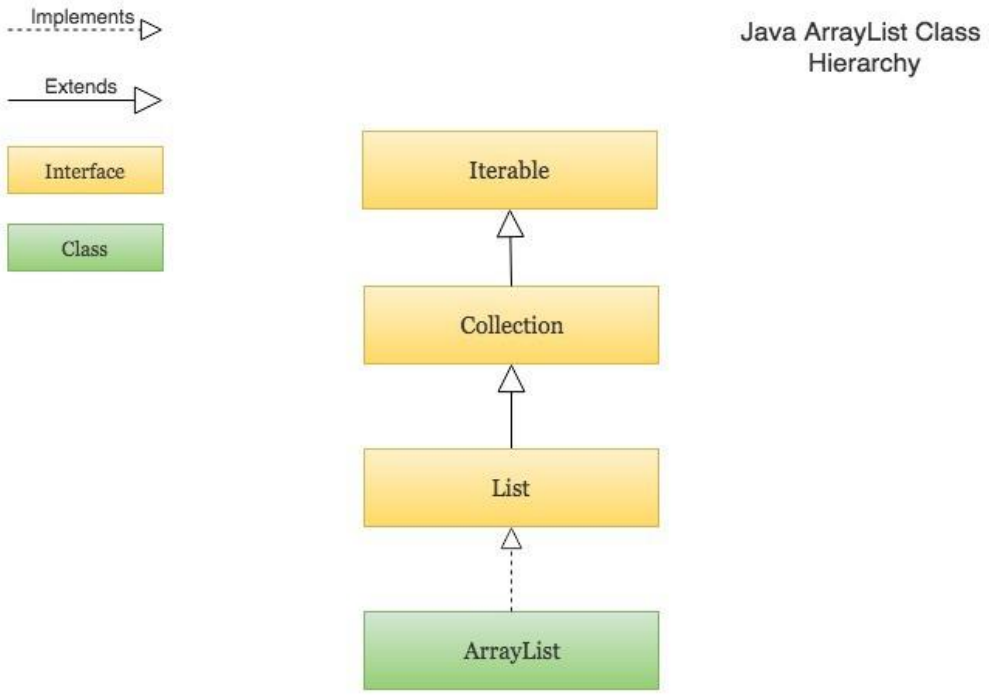
IMPORTANT NOTE:

1. To use any Collection class in your program, you need to import it in your program. It is contained inside java.util package.
2. Whenever you print any Collection class, it gets printed inside the square brackets [].

1. ARRAYLIST CLASS

Simple arrays have fixed size i.e it can store fixed number of elements. But, sometimes you may not know beforehand about the number of elements that you are going to store in your array. In such situations, We can use an ArrayList, which is an array whose size can increase or decrease dynamically.

Following figure shows hierarchy of ArrayList class:-



Following are the properties of array-list class:-

1. ArrayList class extends **AbstractList** class and implements the **List** interface.
2. ArrayList supports dynamic array that can grow as needed. ArrayList has three constructors.
3. **ArrayList**() *//It creates an empty ArrayList*
4. **ArrayList**(*Collection* C) *//It creates an ArrayList that is initialized with elements of the Coll ection C*

5. **ArrayList**(int*capacity*) *//It creates an ArrayList that has the specified initial capacity*
6. ArrayLists are created with an initial size. When this size is exceeded, the size of the ArrayList increases automatically.
7. It can contain Duplicate elements and it also maintains the insertion order.
8. Manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.
9. ArrayLists are not synchronized.
10. ArrayList allows random access because it works on the index basis.

Syntax:

ArrayList< datatype>object =new **ArrayList**< datatype>();

EXAMPLE OF ARRAYLIST

```
import java.util.*
classTest
{
publicstaticvoidmain(String[]args)
{
ArrayList< String> al =newArrayList< String>();
al.add("ab");
al.add("bc");
al.add("cd");
system.out.println(al);
}
}
```

GETTING ARRAY FROM AN ARRAYLIST

toArray() method is used to get an array containing all the contents of the ArrayList. Following are some reasons for why you can need to obtain an array from your ArrayList:

- To obtain faster processing for certain operations.
- To pass an array to methods which do not accept Collection as arguments.
- To integrate and use collections with legacy code.

STORING USER-DEFINED CLASSES

In the above mentioned example we are storing only string object in ArrayList collection. But You can store any type of object, including object of class that you create in Collection classes.

EXAMPLE OF STORING USER-DEFINED OBJECT

Contact class

```
classContact
{
   String first_name;
   String last_name;
   String phone_no;

Public Contact(String fn,Stringln,Stringpn)
{
first_name=fn;
last_name= ln;
phone_no=pn;
}
public String toString()
{
returnfirst_name+" "+last_name+"("+phone_no+")";
}
}
```

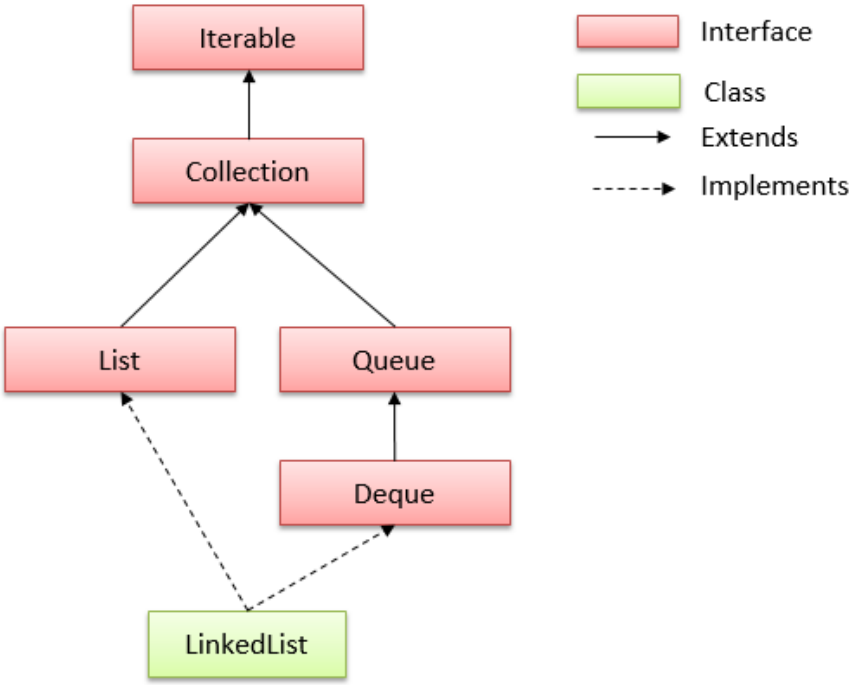
Storing Contact class

```
Public class PhoneBook
{

Public static void main(String[]args)
{
      Contact c1 =newContact("Ricky","Pointing","999100091");
      Contact c2 =newContact("David","Beckham","998392819");
      Contact c3 =newContact("Virat","Kohli","998131319");
```

```
ArrayList< Contact> al =newArrayList< Contact>();
al.add(c1);
al.add(c2);
al.add(c3);
System.out.println(al);
}
}
```

2. LINKEDLIST CLASS

FOLLOWING FIGURE SHOWS HIERARCHY OF LINKEDLIST CLASS:



Following are the properties of array-list class:-

1. LinkedList class extends **AbstractSequentialList** and implements **List**,**Deque** and **Queue**inteface.
2. LinkedList has two constructors.
3. LinkedList()//It creates an empty LinkedList

   LinkedList(Collection C)//It creates a LinkedList that is initialized with elements of the Collection c

4. It can be used as List, stack or Queue as it implements all the related interfaces.
5. They are dynamic in nature i.e it allocates memory when required. Therefore insertion and deletion operations can be easily implemented.
6. It can contain duplicate elements and it is not synchronized.
7. Reverse Traversing is difficult in linked list.
8. In LinkedList, manipulation is fast because no shifting needs to be occurred.

EXAMPLE OF LINKEDLIST CLASS

```
import java.util.* ;
classTest
{
publicstaticvoidmain(String[]args)
{
LinkedList< String>ll=newLinkedList< String>();
ll.add("a");
ll.add("b");
ll.add("c");
ll.addLast("z");
ll.addFirst("A");
System.out.println(ll);
}
}
```

DIFFERENCE BETWEEN ARRAYLIST AND LINKED LIST

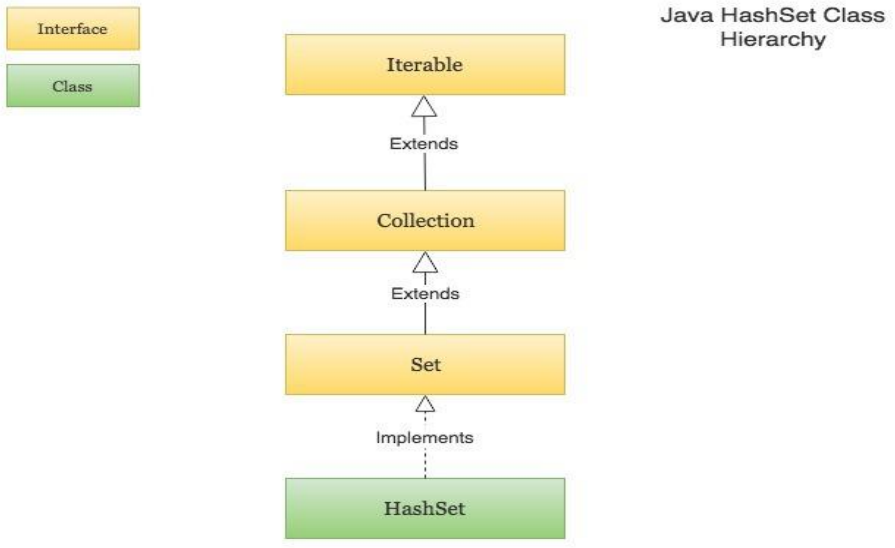
ArrayList and **LinkedList** are the Collection classes, and both of them implements the List interface. The ArrayList class creates the list which is internally stored in a dynamic array that grows or shrinks in size as the elements are added or deleted from it. LinkedList also creates the list which is internally stored in a DoublyLinked List. Both the classes are used to store the elements in the list, but the major difference between both the classes is that ArrayList allows random access to the elements in the list as it operates on an **index-based** data structure. On the other hand, the LinkedList does not allow random access as it does not have indexes to access elements directly, it has to traverse the list to retrieve or access an element from the list.

ArrayList	Linkedlist
ArrayList internally uses a dynamic array to store the elements.	LinkedList internally uses a **doubly linked list** to store the elements.

Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
An ArrayList class can act as a list only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
ArrayList is better for storing and accessing data.	LinkedList is **better for manipulating** data.

- ArrayList extends AbstarctList class whereas LinkedList extends AbstractSequentialList.
- AbstractList implements List interface, thus it can behave as a list only whereas LinkedList implements List, Deque and Queue interface, thus it can behave as a Queue and List both.
- In a list, access to elements is faster in ArrayList as random access is also possible. Access to LinkedList elements is slower as it follows sequential access only.

3. HASHSET CLASS



1. HashSet extends **AbstractSet** class and implements the **Set** interface.
2. HashSet has three constructors.

3. HashSet()//This creates an empty HashSet
4. HashSet(Collection C)//This creates a HashSet that is initialized with the elements of the Collection C
5. HashSet(int capacity)//This creates a HashSet that has the specified initial capacity

1. It creates a collection that uses hash table for storage. A hash table stores information by using a mechanism called **hashing**.
2. 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.
3. HashSet does not maintain any order of elements.
4. HashSet contains only unique elements.

EXAMPLE OF HASHSET CLASS

```
importjava.util.*;
classHashSetDemo
{
publicstaticvoidmain(String args[])
{
HashSet< String>hs=newHashSet< String>();
hs.add("B");
hs.add("A");
hs.add("D");
hs.add("E");
hs.add("C");
hs.add("F");
System.out.println(hs);
}          }
```

4. LINKEDHASHSET CLASS

1. LinkedHashSet class extends **HashSet** class
2. LinkedHashSet maintains a linked list of entries in the set.

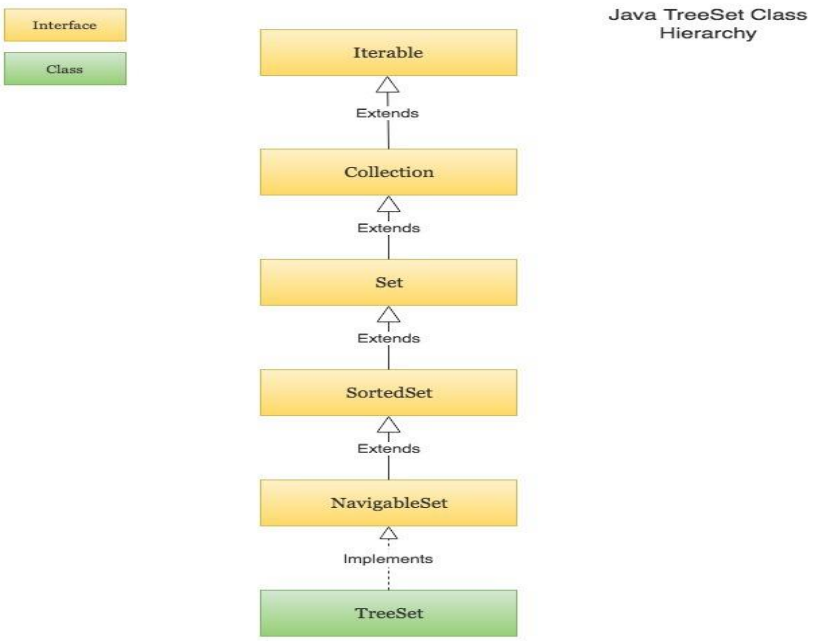
3. LinkedHashSet stores elements in the order in which elements are inserted i.e it maintains the insertion order.

EXAMPLE OF LINKEDHASHSET CLASS

```
importjava.util.*;
classLinkedHashSetDemo
{
publicstaticvoidmain(String args[])
{
LinkedHashSet< String>hs=newLinkedHashSet< String>();
hs.add("B");
hs.add("A");
hs.add("D");
hs.add("E");
hs.add("C");
hs.add("F");
System.out.println(hs);
}
}
```

5. TREESET CLASS

FOLLOWING FIGURE SHOWS HIERARCHY OF TREE-SET CLASS:-



FOLLOWING ARE THE PROPERTIES OF TREESET:-

1. It extends **AbstractSet** class and implements the **NavigableSet** interface.
2. It stores the elements in ascending order.
3. It uses a Tree structure to store elements.
4. It contains unique elements only like HashSet.
5. It's access and retrieval times are quite fast.
6. It has four Constructors.

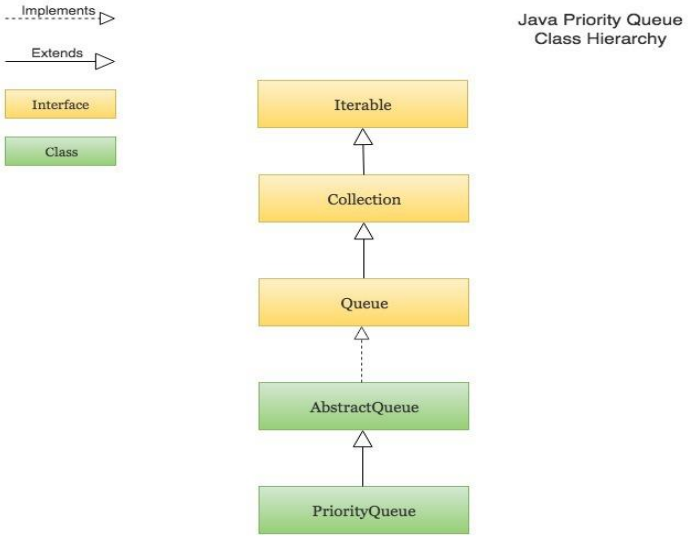
EXAMPLE OF TREESET CLASS

```
importjava.util.*;
classTestCollection11{
publicstaticvoidmain(String args[]){
TreeSet<String> al=newTreeSet<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
  Iterator itr=al.iterator();
```

```
while(itr.hasNext()){
System.out.println(itr.next());
}}}
```

6. PRIORITYQUEUE CLASS

FOLLOWING FIGURE SHOWS HIERARCHY OF TREE-SET CLASS:-



FOLLOWING ARE THE PROPERTIES OF PRIORITYQUEUE:-

1. It extends the **AbstractQueue** class.
2. The PriorityQueue class provides the facility of using queue.
3. It does not orders the elements in FIFO manner.
4. PriorityQueue has six constructors. In all cases, the capacity grows automatically as elements are added.
5. PriorityQueue()//This constructor creates an empty queue. By default, its starting capacity is 11
6. PriorityQueue(int capacity)//This constructor creates a queue that has the specified initial capacity
7. PriorityQueue(int capacity, Comparator comp)//This constructor creates a queue with the specified capacityand comparator
8. //The last three constructors create queues that are initialized with elements of Collection passed in c

9. PriorityQueue(Collection c)
10. PriorityQueue(PriorityQueue c)
11. PriorityQueue(SortedSet c)

Note: If no comparator is specified when a PriorityQueue is constructed, then the default comparator for the type of data stored in the queue is used. The default comparator will order the queue in ascending order. Thus, the head of the queue will be the smallest value. However, by providing a custom comparator, you can specify a different ordering scheme.

Note: Although you can iterate through a PriorityQueue using an iterator, the order of that iteration is undefined. To properly use a PriorityQueue, you must call methods such as offer( ) and poll( ), which are defined by the Queue interface.

EXAMPLE OF PRIORITYQUEUE CLASS

```
importjava.util.*;
classStudyTonight
{
publicstaticvoidmain(String args[])
{
PriorityQueue<String> queue=newPriorityQueue<String>();
queue.add("WE");
queue.add("LOVE");
queue.add("STUDY");
queue.add("TONIGHT");
System.out.println("At head of the queue:"+queue.element());
System.out.println("At head of the queue:"+queue.peek());
System.out.println("Iterating the queue elements:");
  Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
queue.remove();
queue.poll();
System.out.println("After removing two elements:");
  Iterator itr2=queue.iterator();
while(itr2.hasNext()){
```

```
System.out.println(itr2.next());
}}}
```

WAYS TO TRAVERSING A COLLECTION

To access, modify or remove any element from any collection we need to first find the element, for which we have to cycle through the elements of the collection. There are three possible ways to cycle through the elements of any collection.

1. Using Iterator interface
2. Using ListIterator interface
3. Using for-each loop

STEPS TO USE AN ITERATOR

1. Obtain an iterator to the start of the collection by calling the collection's iterator() method.
2. Set up a loop that makes a call to hasNext() method. Make the loop iterate as long as hasNext()method returns true.
3. Within the loop, obtain each element by calling next() method.

ACCESSING ELEMENTS USING ITERATOR

Iterator Interface is used to traverse a list in forward direction, enabling you to remove or modify the elements of the collection. Each collection classes provide **iterator()** method to return an iterator.

Methods of Iterator:

Method	Description
boolean hasNext()	Returns true if there are more elements in the collection. Otherwise, returns false.
E next()	Returns the next element present in the collection. Throws NoSuchElementException if there is not a next element.

void remove()	Removes the current element. Throws IllegalStateException if an attempt is made to call remove() method that is not preceded by a call to next() method.

Example

```
importjava.util.*;
classTest_Iterator
{
publicstaticvoidmain(String[]args)
{
ArrayList< String>ar=newArrayList< String>();
ar.add("ab");
ar.add("bc");
ar.add("cd");
ar.add("de");
   Iterator it =ar.iterator();//Declaring Iterator
while(it.hasNext())
{
System.out.print(it.next()+" ");
}
}
}
```

ACCESSING ELEMENTS USING LISTITERATOR

ListIterator Interface is used to traverse a list in both **forward** and **backward** direction. It is available to only those collections that implements the List Interface.

Methods of ListIterator:

Method	**Description**

void add(E obj)	Inserts obj into the list in front of the element that will be returned by the next call to next() method.
boolean hasNext()	Returns true if there is a next element. Otherwise, returns false.
boolean hasPrevious()	Returns true if there is a previous element. Otherwise, returns false.
E next()	Returns the next element. A NoSuchElementException is thrown if there is not a next element.
int nextIndex()	Returns the index of the next element. If there is not a next element, returns the size of the list.
E previous()	Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.
int previousIndex()	Returns the index of the previous element. If there is not a previous element, returns -1.
void remove()	Removes the current element from the list. An IllegalStateException is thrown if remove() method is called before next() or previous() method is invoked.
void set(E obj)	Assigns obj to the current element. This is the element last returned by a call to either next() or previous() method.

Example:

```
importjava.util.*;
```

```
classTest_Iterator
{
publicstaticvoidmain(String[]args)
{
ArrayList< String>ar=newArrayList< String>();
ar.add("ab");
ar.add("bc");
ar.add("cd");
ar.add("de");
ListIteratorlitr=ar.listIterator();
while(litr.hasNext())//In forward direction
{
System.out.print(litr.next()+" ");
}
while(litr.hasPrevious())//In backward direction
{
System.out.print(litr.previous()+" ");
}
}
}
```

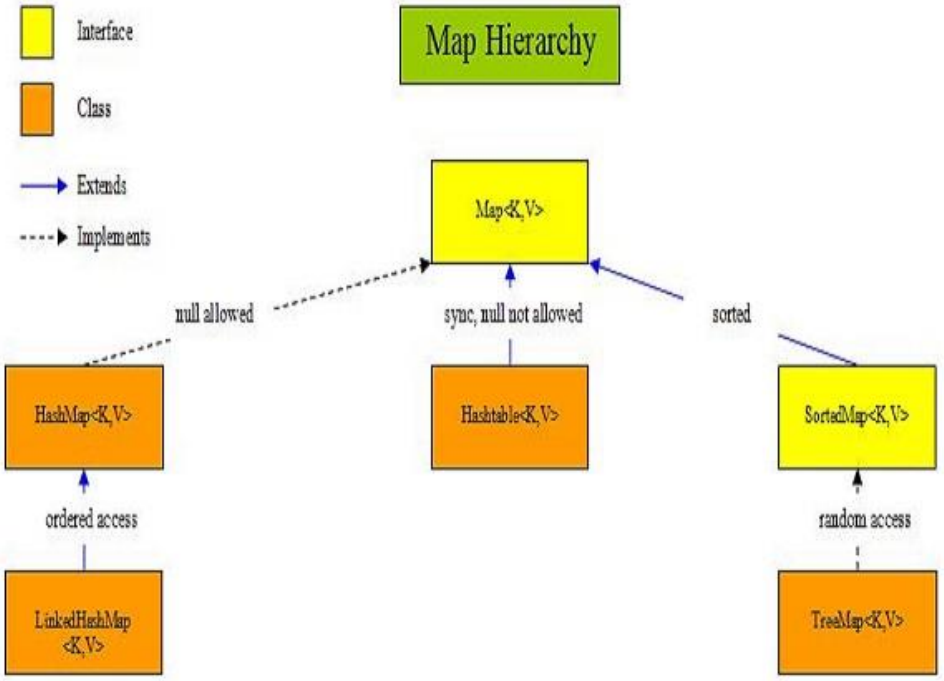
USING FOR-EACH LOOP

for-each version of for loop can also be used for traversing the elements of a collection. But this can only be used if we don't want to modify the contents of a collection and we don't want any **reverse**access. for-each loop can cycle through any collection of object that implements Iterableinterface.

```
importjava.util.*;
classForEachDemo
{
publicstaticvoidmain(String[]args)
{
LinkedList< String> ls =newLinkedList< String>();
ls.add("a");
```

```
ls.add("b");
ls.add("c");
ls.add("d");
for(String str: ls)
{
System.out.print(str+" ");}}}
```

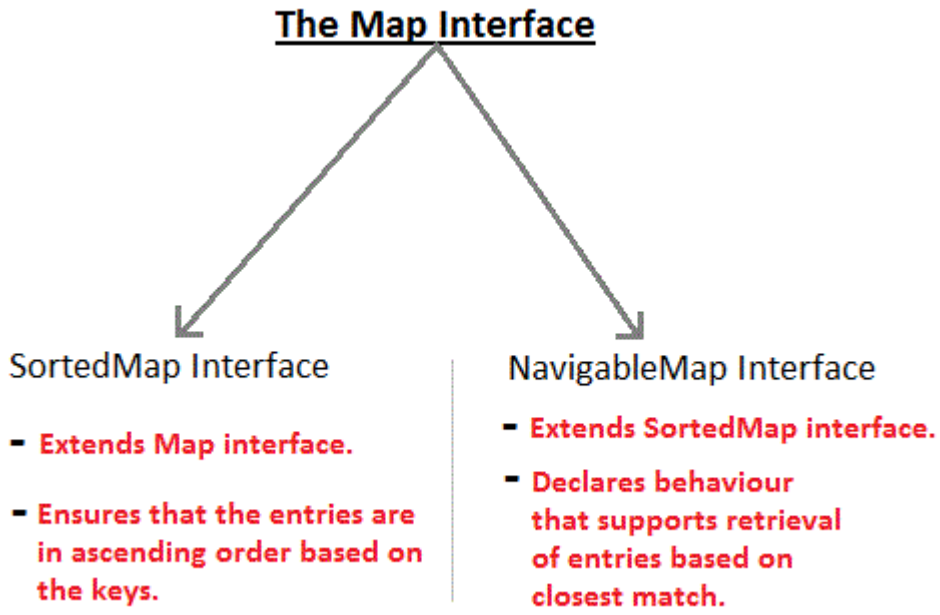
Map Interface



A Map stores data in key and value association. Both key and values are objects. The key must be unique but the values can be duplicate. Although Maps are a part of Collection Framework, they can not actually be called as collections because of some properties that they posses. However we can obtain a **collection-view** of maps.

Interface	Description
Map	Maps unique key to value.
Map.Entry	Describe an element in key and value pair in a map. Entry is sub interface of Map.

NavigableMap	Extends SortedMap to handle the retrienal of entries based on closest match searches
SortedMap	Extends Map so that key are maintained in an ascending order.

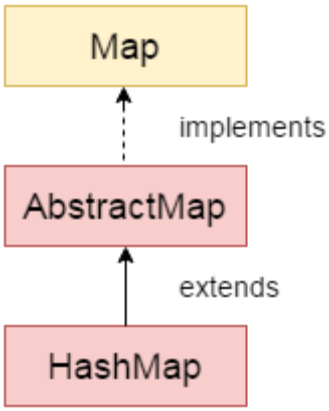


COMMONLY USED METHODS DEFINED BY MAP

- boolean **containsKey**(Object *k*): returns true if map contain *k* as key. Otherwise false.
- Object **get**(Object *k*) : returns values associated with the key *k*.
- Object **put**(Object *k*, Object *v*) : stores an entry in map.
- Object **putAll**(Map *m*) : put all entries from *m* in this map.
- Set **keySet**() : returns **Set** that contains the key in a map.
- Set **entrySet**() : returns **Set** that contains the entries in a map.

1. HASHMAP CLASS

Following figure shows hierarchy of HashMap:



Following are the properties of HashMap Class:

1. HashMap class extends **AbstractMap** and implements **Map** interface.
2. It uses a **hashtable** to store the map. This allows the execution time of get() and put() to remain same.
3. HashMap has four constructor.
4. HashMap()
5. HashMap(Map<?extendsk,?extendsV> m)
6. HashMap(int capacity)
7. HashMap(int capacity,floatfillratio)
8. HashMap does not maintain order of its element.

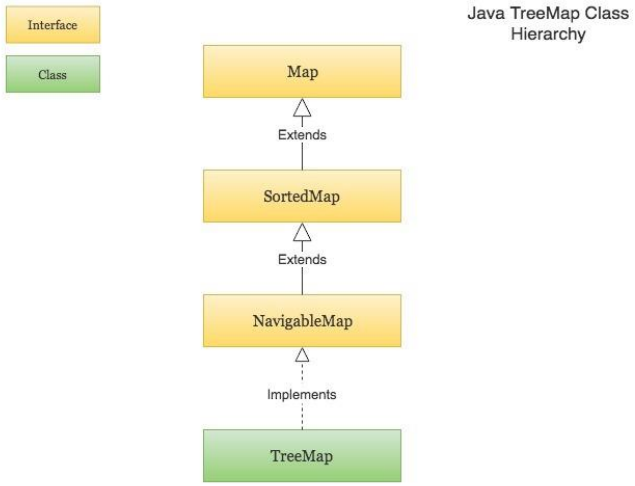
Example:

```
importjava.util.*;
classHashMapDemo
{
publicstaticvoidmain(String args[])
{
HashMap<String,Integer>hm=newHashMap<String,Integer>();
hm.put("a",newInteger(100));
hm.put("b",newInteger(200));
hm.put("c",newInteger(300));
```

```
hm.put("d",newInteger(400));

  Set<Map.Entry<String,Integer>>st=hm.entrySet();//returns Set view
for(Map.Entry<String,Integer>me:st)
{
System.out.print(me.getKey()+":");
System.out.println(me.getValue());
}}}
```

2. TREEMAP CLASS

- Following figure shows hierarchy of TreeMap:



- Following are the properties of TreeMap Class:

1. TreeMap class extends **AbstractMap** and implements **NavigableMap** interface.
2. It creates Map, stored in a tree structure.
3. A **TreeMap** provides an efficient means of storing key/value pair in efficient order.
4. It provides key/value pairs in sorted order and allows rapid retrieval.

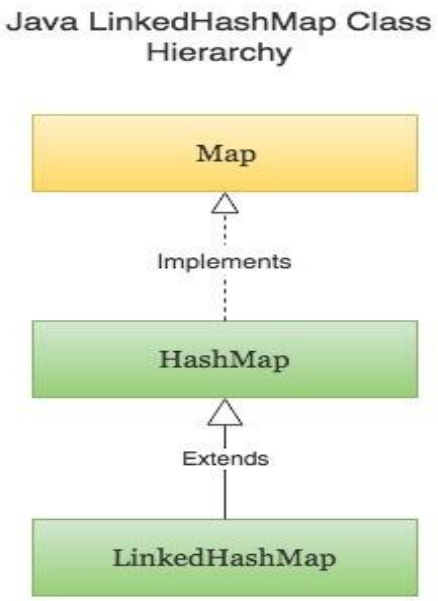
Example:

```
importjava.util.*;
classTreeMapDemo
{
```

```
publicstaticvoidmain(String args[])
{
TreeMap<String,Integer> tm =newTreeMap<String,Integer>();
tm.put("a",newInteger(100));
tm.put("b",newInteger(200));
tm.put("c",newInteger(300));
tm.put("d",newInteger(400));

  Set<Map.Entry<String,Integer>>st=tm.entrySet();
for(Map.Entryme:st)
{
System.out.print(me.getKey()+":");
System.out.println(me.getValue());
}}}
```

3. LINKEDHASHMAP CLASS:

- Following figure shows hierarchy of LinkedHashMap:



- Following are the properties of LinkedHashMap Class:

1. **LinkedHashMap** extends **HashMap** class.
2. It maintains a linked list of entries in map in order in which they are inserted.
3. **LinkedHashMap** defines the following constructor
4. LinkedHashMap()
5. LinkedHashMap(Map<?extendsk,?extendsV> m)
6. LinkedHashMap(int capacity)
7. LinkedHashMap(int capacity,floatfillratio)
8. LinkedHashMap(int capacity,floatfillratio,boolean order)
9. It adds one new method removeEldestEntry(). This method is called by put() and putAll() By default this method does nothing. However we can override this method to remove oldest element in the map. **Syntax**

   protectedbooleanremoveEldestEntry(Map.Entry e)

COMPARATOR INTERFACE

In Java, Comparator interface is used to order(sort) the objects in the collection in your own way. It gives you the ability to decide how elements will be sorted and stored within collection and map.

Comparator Interface defines compare() method. This method has two parameters. This method compares the two objects passed in the parameter. It returns 0 if two objects are equal. It returns a positive value if object1 is greater than object2. Otherwise a negative value is returned. The method can throw a **ClassCastException** if the type of object are not compatible for comparison.

RULES FOR USING COMPARATOR INTERFACE

1. If you want to sort the elements of a collection, you need to implement Comparator interface.
2. If you do not specify the type of the object in your Comparator interface, then, by default, it assumes that you are going to sort the objects of type Object. Thus, when you override the compare() method ,you will need to specify the type of the parameter as Object only.
3. If you want to sort the user-defined type elements, then while implementing the Comparator interface, you need to specify the user-defined type generically. If you do not specify the

user-defined type while implementing the interface,then by default, it assumes Object type and you will not be able to compare the user-defined type elements in the collection

For Example:

If you want to sort the elements according to roll number, defined inside the class Student, then while implementing the Comparator interface, you need to mention it generically as follows:

```
classMyComparatorimplementsComparator<Student>{}
```

If you write only,classMyComparatorimplementsComparator{}

Then it assumes, by default, data type of the compare() method's parameter to be Object, and hence you will not be able to compare the Student type(user-defined type) objects.

Student class

```
classStudent
int roll;
  String name;
Student(intr,String n)
{
     roll = r;
     name = n;
}
public String toString()
{
return roll+" "+name;
}
```

MyComparator class

This class defines the comparison logic for Student class based on their roll. Student object will be sorted in ascending order of their roll.

```
classMyComparatorimplementsComparator<Student>
{
publicintcompare(Student s1,Student s2)
{
if(s1.roll == s2.roll)return0;
elseif(s1.roll > s2.roll)return1;
elsereturn-1;
```

```
}
}
publicclassTest
{
publicstaticvoidmain(String[]args)
{
TreeSet< Student>ts=newTreeSet< Student>(newMyComparator());
ts.add(newStudent(45,"Rahul"));
ts.add(newStudent(11,"Adam"));
ts.add(newStudent(19,"Alex"));
System.out.println(ts);
}}
```

Note:

- When we are sorting elements in a collection using Comparator interface, we need to pass the class object that implements Comparator interface.
- To sort a TreeSet collection, this object needs to be passed in the constructor of TreeSet.
- If any other collection, like ArrayList,was used, then we need to call sort method of Collections class and pass the name of the collection and this object as a parameter.
- For example, If the above program used ArrayList collection, the public class test would be as follows:

```
publicclassTest
{publicstaticvoidmain(String[]args){
ArrayList< Student>ts=newArrayList< Student>();
ts.add(newStudent(45,"Rahul"));
ts.add(newStudent(11,"Adam"));
ts.add(newStudent(19,"Alex"));
Collections.sort(ts,newMyComparator());/*passing the name of the ArrayList and the
object of the class that implements Comparator in a predefined sort() method in Collections
class*/
System.out.println(ts);
```

```
}        }
```

LEGACY CLASSES

Early version of java did not include the Collections framework. It only defined several classes and interfaces that provide methods for storing objects. When Collections framework were added in J2SE 1.2, the original classes were reengineered to support the collection interface. These classes are also known as Legacy classes. All legacy classes and interface were redesign by JDK 5 to support Generics. In general, the legacy classes are supported because there is still some code that uses them.

The following are the legacy classes defined by **java.util** package

1. Dictionary
2. HashTable
3. Properties
4. Stack
5. Vector

There is only one legacy interface called **Enumeration**

NOTE: All the legacy classes are synchronized

HASHTABLE CLASS

1. Like HashMap, Hashtable also stores key/value pair. However neither **keys** nor **values** can be **null**.
2. There is one more difference between **HashMap** and **Hashtable** that is Hashtable is synchronized while HashMap is not.
3. Hashtable has following four constructor
    1. Hashtable()//This is the default constructor. The default size is 11.
    2. Hashtable(int size)//This creates a hash table that has an initial size specified by size.
    3. Hashtable(int size,floatfillratio)//This creates a hash table that has an initial size specified by sizeand a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how fullthe hash table can be before it is resized upward. Specifically, when the number of elements is greaterthan the capacity of the hash table multiplied by its fill ratio, the hash table is expanded.If you do not specify a fill ratio, then 0.75 is used.
    4. Hashtable(Map<?extendsK,?extendsV> m)//This creates a hash table that is initialized with theelements in m. The capacity of the hash table is set to twice the number of elements in m.The default load factor of 0.75 is used.

EXAMPLE OF HASHTABLE

```
importjava.util.*;
classHashTableDemo
{
publicstaticvoidmain(String args[])
{
Hashtable<String,Integer>ht=newHashtable<String,Integer>();
ht.put("a",newInteger(100));
ht.put("b",newInteger(200));
ht.put("c",newInteger(300));
ht.put("d",newInteger(400));
  Set st=ht.entrySet();
  Iterator itr=st.iterator();
while(itr.hasNext())
{
Map.Entry m=(Map.Entry)itr.next();
System.out.println(itr.getKey()+" "+itr.getValue());
}
}}
```

Difference between important collection classes:

DIFFERENCE BETWEEN ARRAYLIST AND LINKED LIST

ArrayList	LinkedList
ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.
ArrayList is not efficient for manipulation because too much is required.	LinkedList is efficient for manipulation.

ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
ArrayList provides random access.	LinkedList does not provide random access.
ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

DIFFERENCE BETWEEN LIST AND LINKED SET

List	Set
List is an ordered collection it maintains the insertion order, which means upon displaying the list content it will display the elements in the same order in which they got inserted into the list.	Set is an unordered collection, it doesn't maintain any order. There are few implementations of Set which maintains the order such as LinkedHashSet (It maintains the elements in insertion order).
List allows duplicates while Set doesn't allow duplicate elements.	All the elements of a Set should be unique if you try to insert the duplicate element in Set it would replace the existing value.
List implementations: ArrayList, LinkedList etc.	Set implementations: HashSet, LinkedHashSet, TreeSet etc.
List allows any number of null values.	Set can have only a single null value at most.

DIFFERENCE BETWEEN HASHSET AND HASHMAP

HashSet	HashMap
HashSet class implements the Set interface	HashMap class implements the Map interface
HashSet does not allow duplicate elements that means you can not store duplicate values in HashSet.	HashMap does not allow duplicate keys however it allows to have duplicate values.
HashSet permits to have a single null value.	HashMap permits single null key and any number of null values.

DIFFERENCE BETWEEN HASHMAP AND HASHTABLE

Hashtable	HashMap
Hashtable class is synchronized.	HashMap is not synchronized.
Because of Thread-safe, Hashtable is slower than HashMap	HashMap works faster.
Neither **key** nor **values** can be null	Both **key** and **values** can be null
Order of table remain constant over time.	does not guarantee that order of map will remain constant over time.

CHAPTER 3 JAVA JDBC

INTRODUCTION TO JDBC CONCEPT

JDBC (Java Database Connectivity) is uses for connect java application with database. It is Java SE(Standard Edition) technology, which is install automatically with the JDK software. JDBC is an API (Application programming interface) used to communicate Java application to database in database independent and platform independent manner. It provides classes and interfaces to connect or communicate Java application with database.

JDBC in Java

Jdbc ia an API (Application programming interface) used to communicate Java application to database in database independent and platform independent manner. It provides classes and interfaces to connect or communicate Java application with database.

Jdbc ia a part of JDK software so no need to install separate software for Jdbc API

Jdbc API consists of two packages

1. java.sql package
2. javax.sql package

What is JDBC?

JDBC (Java Database Connectivity) is uses for connect java application with database. JDBC API is a Java API that can access any kind of data stored in a Relational Database. It enables Java programs to execute SQL statements. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX.

Before reading JDBC you need basic knowledge of Core java and most important some topics are required like Abstract, Interface, Exception Handling, Collection Framework etc.

Why use JDBC?

In earlier days for communicate front end application to database, front-end application used a set of function given by database vendor, to connect with a database.

But problem with the above communication is a front end application become as a database dependent application because every database vendor give its own set of function for communication.

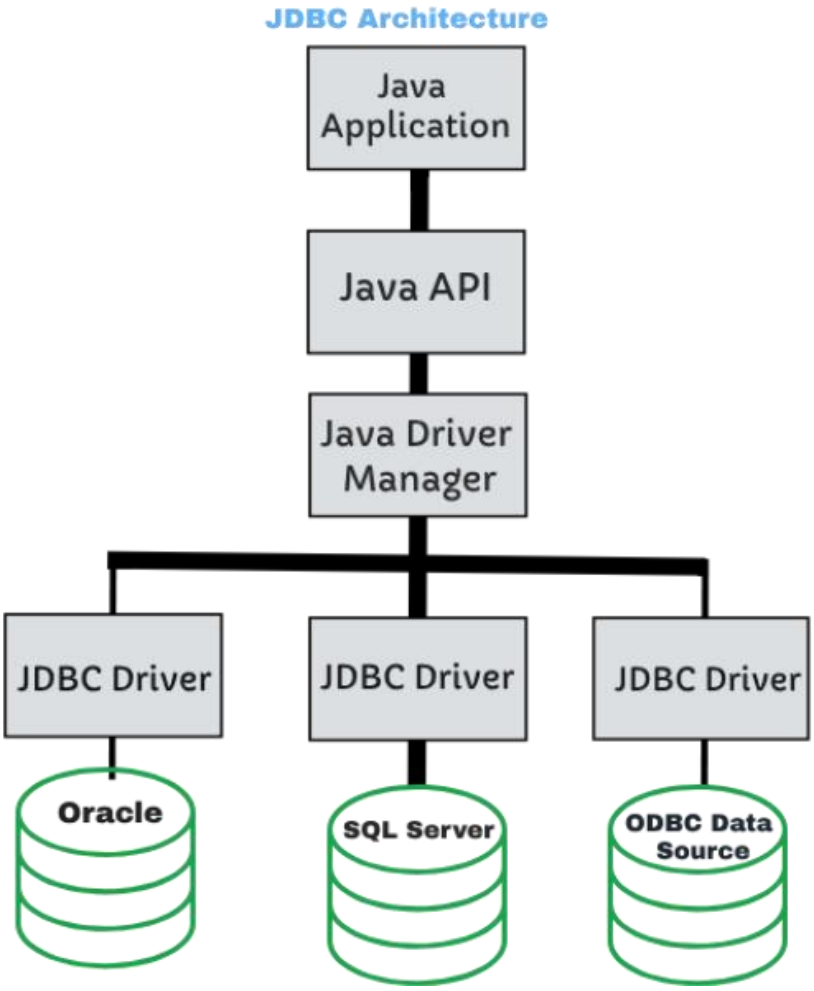
To overcome the database dependent problem ODBC (Open database connectivity) community formed by Microsoft with Simba technologies.

ODBC community has provided ODBC API, to connect with any database in a database independent manner.

Why ODBC not use in Java Application ?

ODBC API is written in C language with pointer but Java application does not contain pointer so internally non pointers java code is converted to C pointers code this conversion is a time-consuming process so the connectivity is very slow.

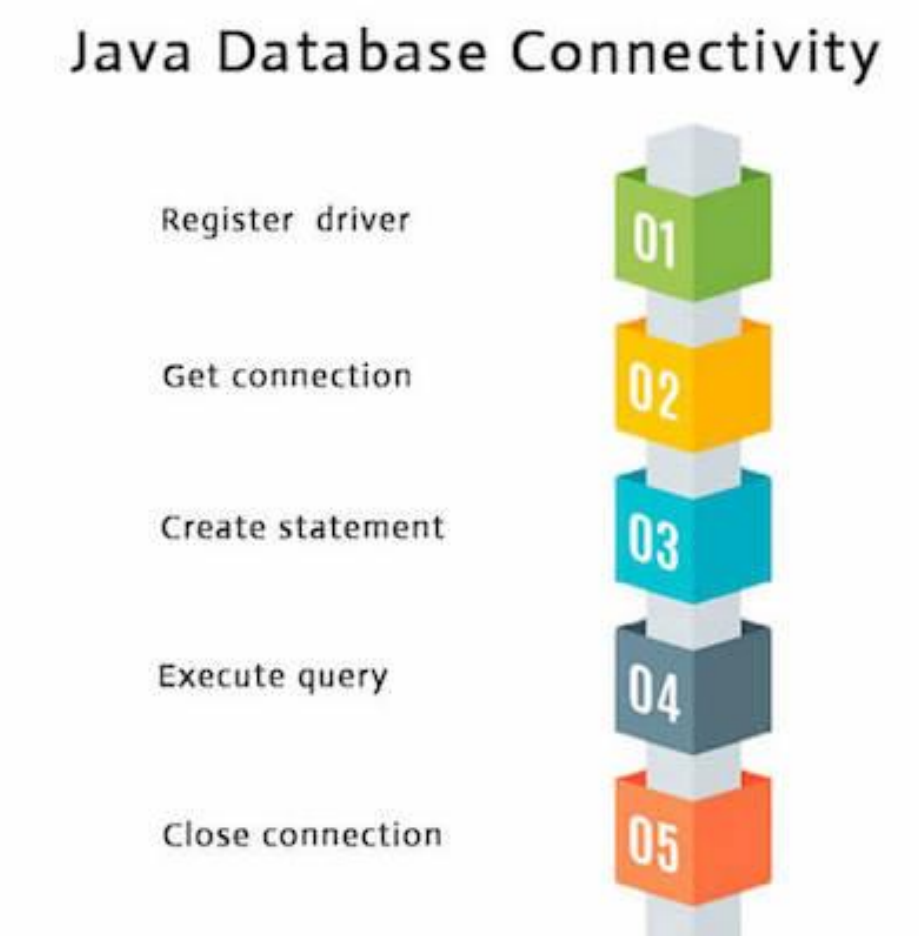
Java application is platform independent but if it is combined with ODBC then it become platform dependent but this is against of java motto or principal. To solved the above problems, Sum MicroSystem introduced JDBC technology. Jdbc technology makes java applications as platform independent and database independent.



Difference between ODBC and JDBC

ODBC	JDBC
Odbc is platform dependent and database independent.	Jdbc is both platform and database independent.
Odbc implemented in C language with pointer	Jdbc implemented in java without pointer

Procedure to connect java with any database



Above figure shows the steps included in connecting java application with any database.

More precisely these are divided into following different steps:

1. Load the JDBC driver class or register the JDBC driver.
2. Establish the connection
3. Create a statement
4. Execute the sql commands on database and get the result
5. Print the result
6. Close the connection

1. Register the driver class

In this step we load the JDBC driver class into JVM. This step is also called as registering the JDBC driver. The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class. This step can be completed in two ways.

- class.forName("fully qualified classname")
- DriveManager.registerDriver(object of driver class)

Syntax of forName() method

```
publicstaticvoid forName(String className)throws ClassNotFoundException
```

Sun.Jdbc.Odbc.JdbcOdbcDriver is a driver class provided by Sun MicroSystem and it can be loaded into jvm like the following.

Syntax

```
class.forName("Sun.Jdbc.Odbc.JdbcOdbcDriver");
```

Syntax

```
Sun.Jdbc.Odbc.JdbcOdbcDriver jod=new Sun.Jdbc.Odbc.JdbcOdbcDriver();
DriverManager.registerDriver(jod);
```

2. Create the connection object

In this step connection between a java program and a database will be opened. To open the connection, we call getConnection() method of DriverManager class.
For getConnection() method we need to pass three parameters.

1. url
2. username
3. password

url: url is used to select one register JDBC driver among multiple registered driver by DriverManager class.

username and password: username and password are used for authentication purpose.

Syntax of getConnection() method

```
1) publicstatic Connection getConnection(String url)throws SQLException
2) publicstatic Connection getConnection(String url,String name,String password)
throws SQLException
```

Example to establish connection with the Oracle database

```
Connection con=new DriverManager.getConnection(url, username, password);
Example:
Connection con=new DriverManager.getConnection(Jdbc:Odbc:< dsn >", "scott","tiger");
```

3. Create the Statement object :

To transfer sql commands from java program to database we need statement object. To create a statement object we call **createStatement()** method of connection interface.The object of statement is responsible to execute **queries** with the database.

Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
Statement stmt=new createStatement();
```

4. Executing queries

Call any one of the following three methods of Statement interface is used to execute queries to the database and to get the output.

- **executeUpdate():** Used for non-select operations.
- **executeQuery():** Used for select operation.
- **execute():** Used for both select or non-select operation.

5. Print the result.

```
System.out.println(output);
```

6.Closing connection : Close the connection.

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
publicvoid close()throws SQLException
```

Example for close connection

```
con.close();
```

Example

```
import java.sql.*;
class CreateTable
{
publicstaticvoid main(String[] args) throws Exception
{
//step-1
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
System.out.println("driver is loaded");
//step-2
Connection con=DriverManager.getConnection("jdbc:odbc:ramadsn","system","system");
System.out.println("connection is established");
//step-3
Statement stmt=con.createStatement();
System.out.println("statement object is cretaed");
//step-4
```

```
int i=stmt.executeUpdate("create table student(sid number(3),sname varchar2(10),marks number(5))");
//step-5
System.out.println("Result is="+i);
System.out.println("table is created");
//step-6
stmt.close();
con .close();
}
}
```

JDBC driver

Jdbc API contains a set of classes and Interfaces where classes definition is provided by Sun Micro System and Interfaces Implementation are not provided. For interface Implementation classes will be provided by vendors, these set of classes are called **Jdbc Driver Software**.

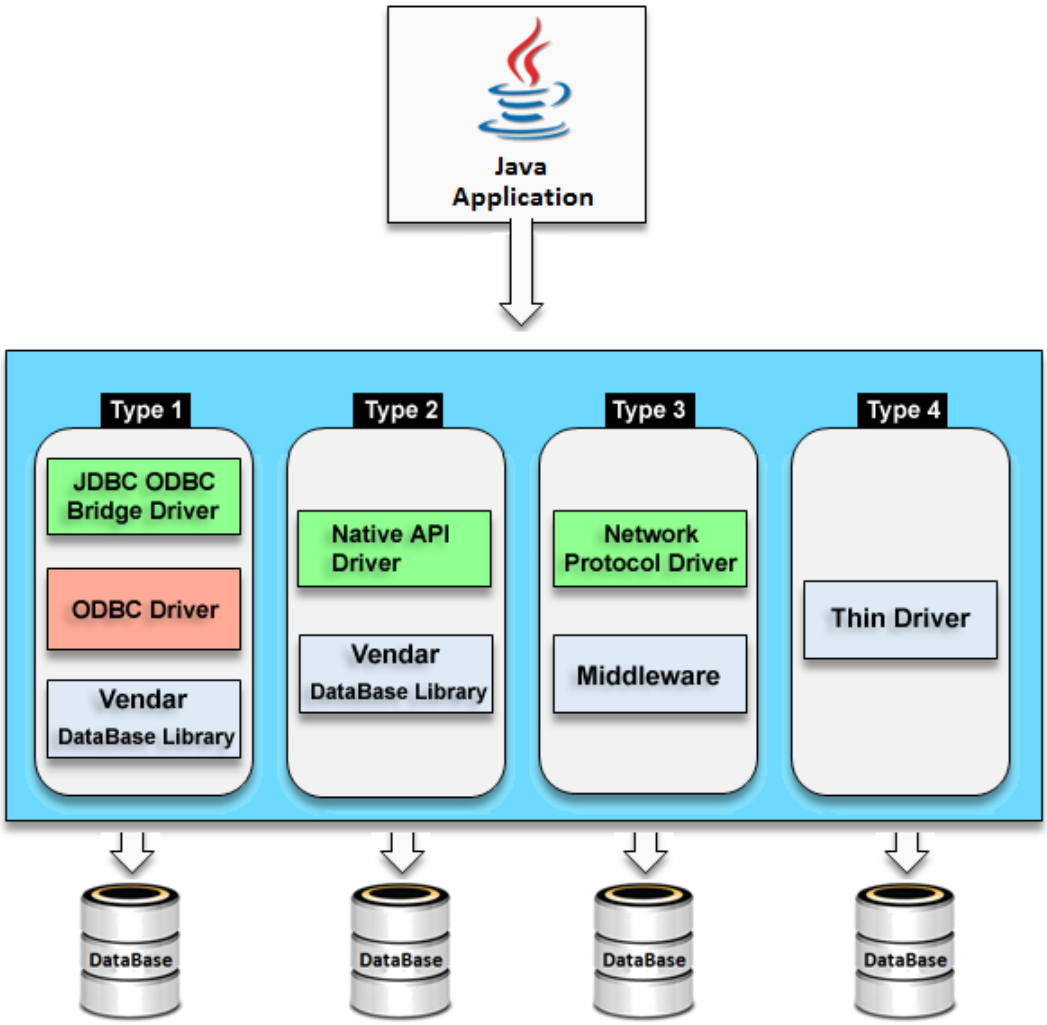
A Driver software contains set of classes among these classes one class is driver class.Driver class is a mediator class between a java application and a database.

Java Application use Jdbc API and this API connect to driver class, then driver class connect to database.

Types of Jdbc Driver

JDBC Driver is a software component that enables java application to communicate with the database.There are 4 types of JDBC drivers, they are:

- Jdbc-Odbc Bridge Driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

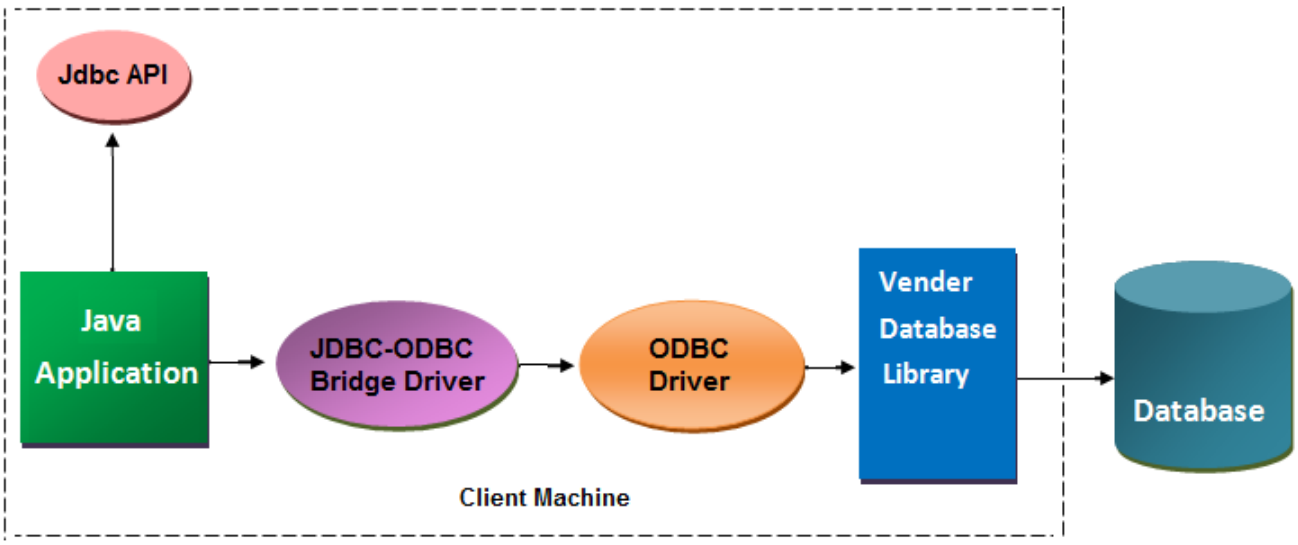


Short Description of Jdbc drivers

Driver	Database	Platform
Type – 1	Independent	Dependent.
Type – 2	Dependent	Dependent.
Type – 3	Independent	Independent.
Type – 4	Dependent	Independent.

JDBC ODBC BRIDGE DRIVER

This driver connect a java program with a database using Odbc driver. It is install automatically along with JDK software. It is provided by Sun MicroSystem for testing purpose this driver can not be used in real time application. This driver convert JDBC calls into Odbc calls(function) So this is called a **bridge driver**.



JDBC-ODBC bridge driver

Advantage of bridge driver

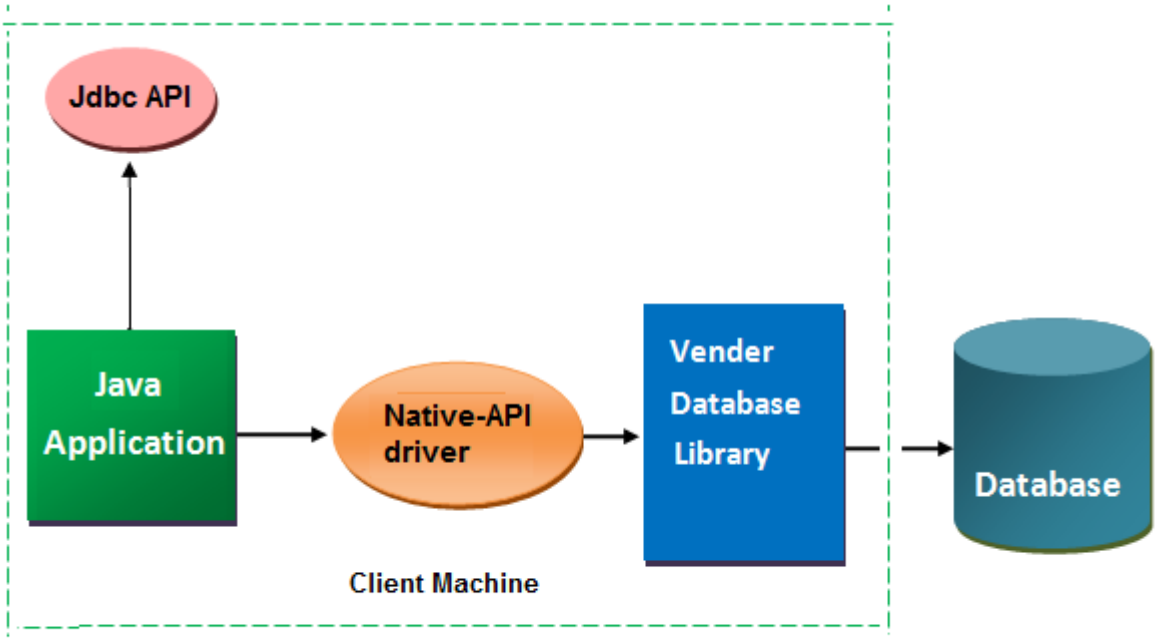
- Easy to use
- It is a database independent driver
- Can be easily connected to any database.
- This driver software is built-in with JDK so no need to install separately.

Disadvantage of bridge driver

- It is a slow driver so not used in real time application
- Because of Odbc connectivity it is a platform dependent driver.
- It is not a portable driver.
- It is not suitable for applet to connect with database.

NATIVE API DRIVER

Native API driver uses native API to connect a java program directly to the database. Native API id s C, C++ library, which contains a set of function used to connect with database directly. Native API will be different from one database to another database. So this Native API driver is a database dependent driver.



Native-API Driver

Advantage of Thin driver

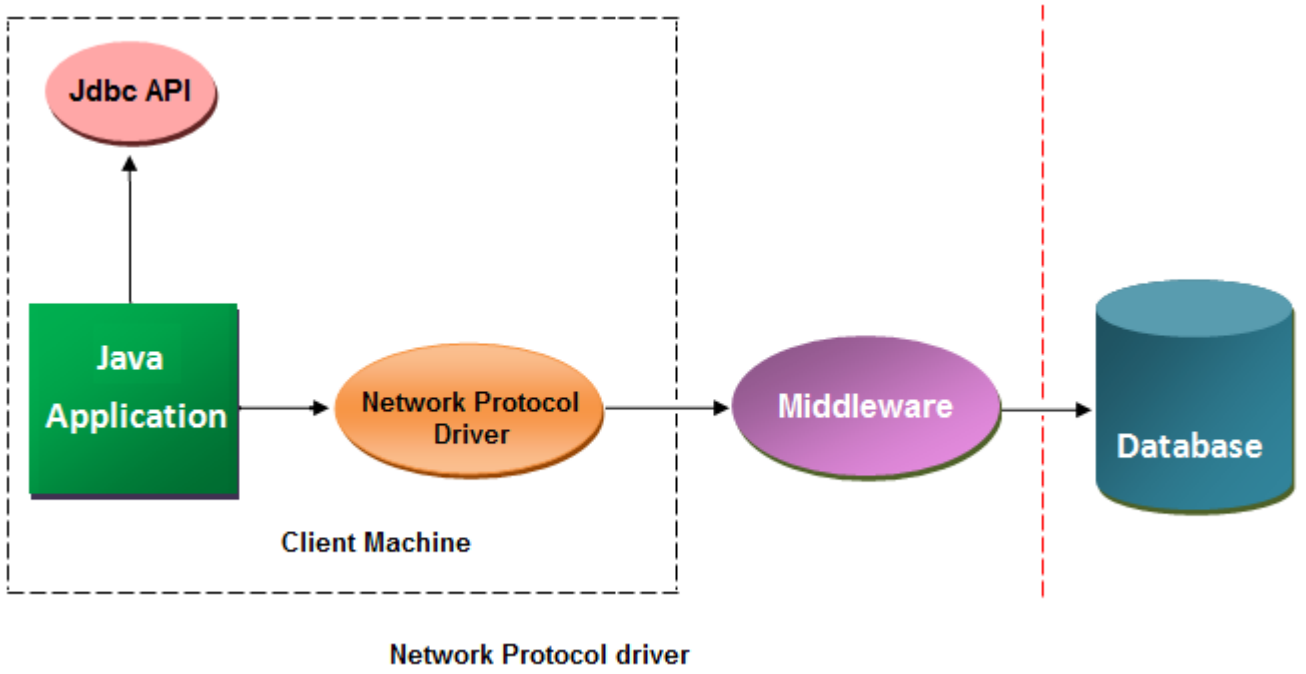
- Native API driver comparatively faster than JDBC-ODBC bridge driver.

Disadvantage of Thin driver

- Native API driver is database dependent and also platform dependent because of Native API.

NETWORK PROTOCOL DRIVER IN JDBC

The Network Protocol driver uses middle-ware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java..



Network Protocol driver

Advantage of Network Protocol driver

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- This driver is both database and platform independent driver

Disadvantage of Network Protocol driver

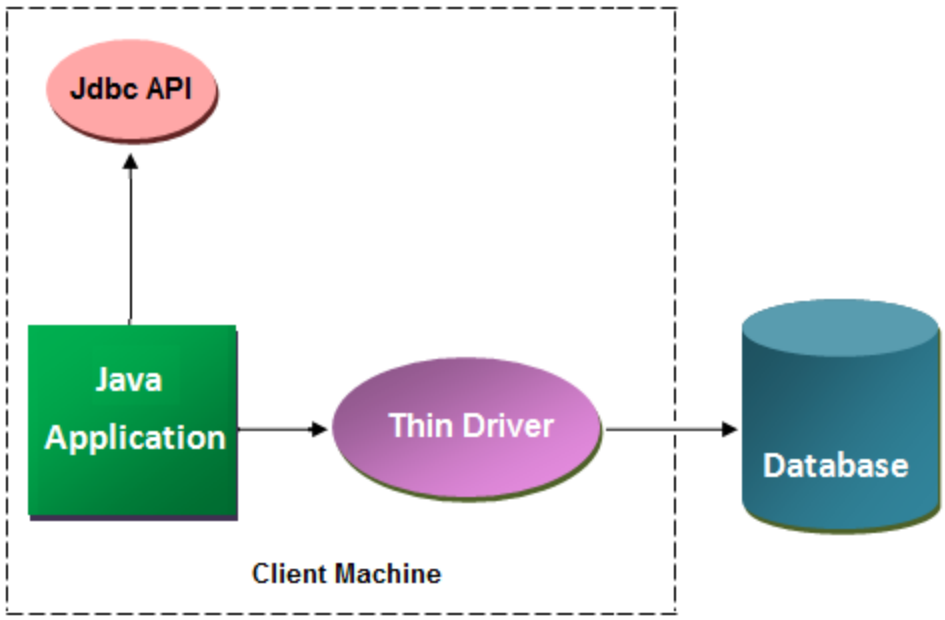
- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

THIN DRIVER IN JDBC

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

This thin driver uses the following three information to connect with a database.

- Ip address of a machine (system), where the database server is running.
- Port number of the database server.
- Database name, also called SID (service ID).



Thin driver

Advantage of Thin driver

- Thin driver is the fastest driver among all Jdbc drivers.
- No software is required at client side or server side.
- It is portable driver because it is platform independent.
- It can be used to connect an applet with the database.

Disadvantage of Thin driver

- Thin driver is a database dependent driver.

Why thin driver is database dependent driver ?

Because thin driver internally uses native protocol. Native protocol is a server dependent protocol it means the protocol can establish connection with a particular server only.

Thin driver connect with database

Oracle corporation has provided two JDBC driver software for connection java application to a database of oracle server.

- Oracle oci driver.
- Oracle thin driver.

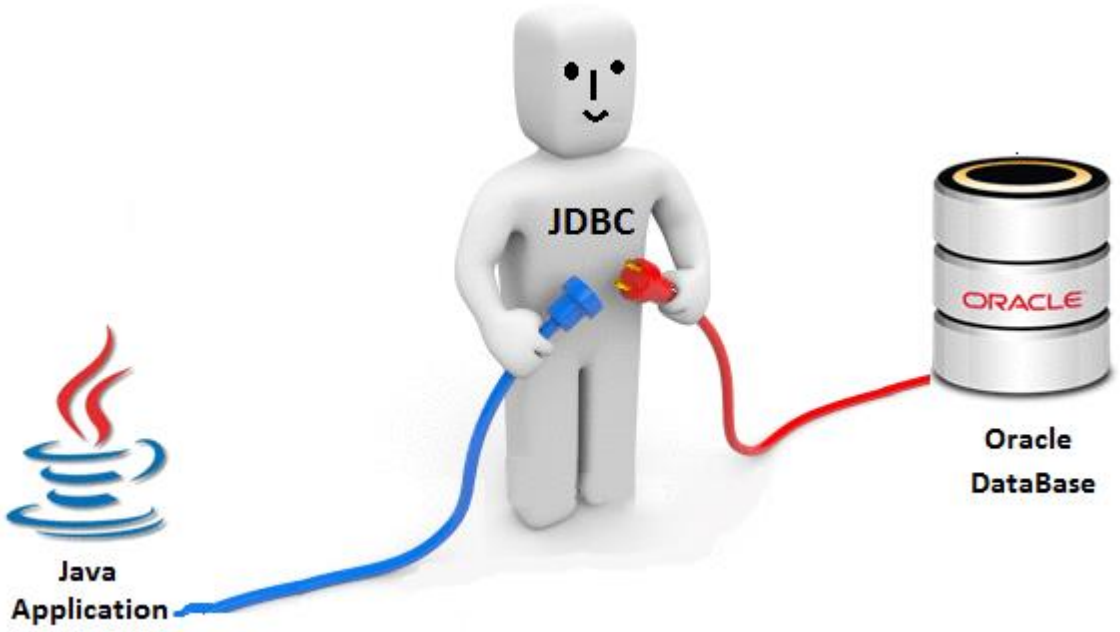
The following are the connection properties of oracle thin driver.

driver name:	Oracle.Jdbc.OracleDriver
url:	Jdbc:Oracle:thin@ipaddress:sid
Name	System
Password	Tiger

How to Connect Java Application with Oracle Database

Oracle corporation has provided two Jdbc driver software for connection java application to a database of oracle server.

- Oracle oci Driver
- Oracle thin Driver



For connecting java application with the oracle database, we need to follow below steps. In this example we are using Oracle 10g as the database. So we need to know following information for the oracle database.

The following are the connection properties of oracle thin driver.

Driver Class:	Oracle.Jdbc.OracleDriver
Connection url:	jdbc:oracle:thin:@localhost:1521:xe
Username:	The default username for the oracle database is system.
Password	Password is given by the user at the time of installing the oracle database.

jdbc:oracle:thin:@localhost:1521:x

- **jdbc** is the API
- **oracle** is the database
- **thin** is the driver
- **localhost** is the server name on which oracle is running, we may also use IP address
- **1521** is the port number and
- **XE** is the Oracle service name.

Note: You may get all above information from the tnsnames.ora file.

Create a table in oracle database

```
create table student(roll number(10),name varchar2(40));
```

Example to Connect Java Application with Oracle database

```
import java.sql.*;
class OracleCon
{
publicstaticvoid main(String args[]){
try
{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from student");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();
}
catch(Exception e)
{
 System.out.println(e);
}
}
}
```

Connection Interface in JDBC

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(),rollback() etc**Method of Connection Interface**

	method	Discription
1	public Statement createStatement()	creates a statement object that can be used to execute SQL queries.

2	public Statement createStatement(int resultSetType,int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3	public void setAutoCommit(boolean status)	Used to set the commit status.By default it is true.
4	public void commit()	saves the changes made since the previous commit/rollback permanent.
5	public void rollback()	Drops all changes made since the previous commit/rollback.
6	public void close():	closes the connection and Releases a JDBC resources immediately.

Drivermanager Class of JDBC

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Methods of DriverManager Class

	method	Description
1	public static void registerDriver(Driver driver)	Used to register the given driver with DriverManager.
2	public static void deregisterDriver(Driver driver):	Used to registered the given driver (drop the driver from the list) with DriverManager.

3	public static Connection getConnection(String url):	Used to establish the connection with the specified url.
4	public static Connection getConnection(String url,String userName,String password):	Used to establish the connection with the specified url, username and password.

Statement Interface in JDBC

The Statement interface provides methods to execute queries with the database.

Method of Statement Interface

	method	Description
1	public ResultSet executeQuery(String sql)	Used to execute SELECT query. It returns the object of ResultSet.
2	public int executeUpdate(String sql)	Used to execute specified query, it may be create, drop, insert, update, delete etc.
3	public boolean execute(String sql)	Used to execute queries that may return multiple results.
4	public int[] executeBatch()	Used to execute batch of commands.

ResultSet Interface in JDBC

The object of ResultSet maintains a cursor pointing to a particular row of data. Initially, cursor points to before the first row.

Method of ResultSet Interface

	method	Description

1	public boolean next()	Used to move the cursor to the one row next from the current position.
2	public boolean previous()	Used to move the cursor to the one row previous from the current position.
3	public boolean first()	Used to move the cursor to the first row in result set object.
4	public boolean last()	Used to move the cursor to the last row in result set object.
5	public boolean last()	Used to move the cursor to the last row in result set object.

Scrollable Resultset in JDBC

In Jdbc ResultSet Interface are classified into two type;.

- **Non-Scrollable ResultSet in JDBC**
- **Scrollable ResultSet**

By default a ResultSet Interface is Non-Scrollable, In non-scrollable ResultSet we can move only in forward direction (that means from first record to last record), but not in Backward Direction, If you want to move in backward direction use **Scrollable Interface**.

Difference between Scrollable ResultSet and Non-Scrollable ResultSet

	Non-Scrollable ResultSet	Scrollable ResultSet
1	Cursor move only in forward direction	Cursor can move both forward and backward direction
1	Slow performance, If we want to move nth record then we need to n+1 iteration	Fast performance, directly move on any record.

1	Non-Scrollable ResultSet cursor can not move randomly	Scrollable ResultSet cursor can move randomly

Methods of Scrollable ResultSet

Below all methods are used for move the cursor in Scrollable ResultSet.

- **afterLast** Used to move the cursor after last row.
- **BeforeFirst:** Used to move the cursor before first row.
- **previous:** Used to move the cursor backward.
- **first:** Used to move the cursor first at row.
- **last:** Used to move the cursor at last row.

Example of Scrollable ResultSet

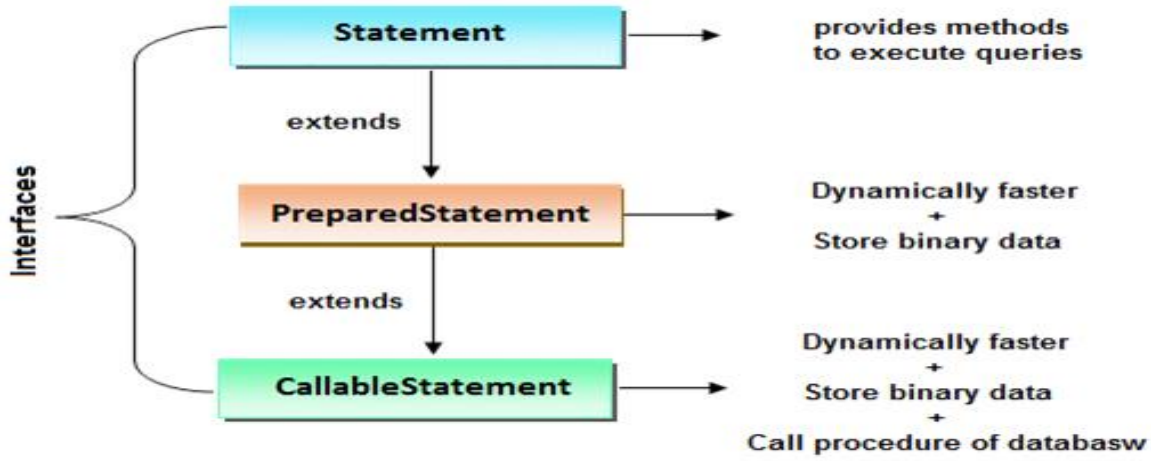
```
import java.sql.*;
class ScrollableTest
{
publicstaticvoid main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@John-pc:1521:xe","system","system");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stmt.executeQuery("select * from student");

//reading from button to top
rs.afterLast();
while(rs.previous())
{
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
}
```

```
//move the cursor to 3rd record
rs.absolute(3);
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));

//move the cursor to 2nd record using relative()
rs.relative(-1);
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3));
int i=rs.getRow(); // get cursor position
System.out.println("cursor position="+i);

//cleanup
rs.close();
stmt.close();
con.close();
 }
}
```

Difference between PreparedStatement and Statement

If the sql command is same then actually no need to compiling it for each time before it is executed. So the performance of an application will be Increased. In this case **PreparedStatement** is used.

PreparedStatement Interface is derived Interface of statement and CallableStatement is derived Interface of PreparedStatement.



Why use PreparedStatement

We know that when working with **Statement Interface** of JDBC the sql command will be compiled first and then it is executed at database side even through the same sql command is execute repeatedly but each time the command is compiled and then executed at database. Due to this performance of application will be decreased So overcome this problem use PreparedStatement. In PreparedStatement, If the sql command is same then actually no need to compiling it for each time before it is executed.

In case of preparedStatement

- First sql command is send to database for compilation and then compiled code will be stored in preparedStatement object.
- The compiled code will be executed for n number of time without recompiling the sql command.
- The criteria to use preparedStatement is when we want to execute same sql query for multiple times with different set of values.
- Comparatively preparedStatement is faster than Statement Interface.

Difference between PreparedStatement and Statement

	Statement	PreparedStatement
1	Statement interface is slow because it compile the program for each execution	PreparedStatement interface is faster, because its compile the command for once.

2	We can not use ? symbol in sql command so setting dynamic value into the command is complex	We can use ? symbol in sql command, so setting dynamic value is simple.
3	We can not use statement for writing or reading binary data (picture)	We can use PreparedStatement for reading or writing binary data.

Create an object of PreparedStatement

Syntax

```
Connection con; // con is reference of connection
PreparedStatement pstmt=con.prepareStatement("sql command");
```

Why use '?' symbol in PreparedStatement

To pre-compile a command only syntax of the command is required so we can use '?' symbol for value in the command. '?' symbol is called parameter or replacement operator or place-resolution operator.

Syntax

```
PreparedStatement pstmt=con.prepareStatement("Insert into student_table value(?, ?, ?)");
```

Note: In PreparedStatement only '?' symbol are allow, no other symbols are allowed.

Note: '?' is only for replacing value but not for table name or column names.

Note: '?' symbol are not allowed in DDL operation.

Setting value

We call setxxx() methods to set the value in place of ? symbols, before executing the command. Here pass two parameters for setxxx(), where first parameter is index and second is value. XXX means any data type.

Syntax

```
pstmt.setInt(1,102);
```

Example of PreparedStatement

```
import java.sql.*;
import javax.sql.*;//PreparedStatement;
import java.util.*;
class PrepardTest1
{
Connection con;
void openConnection()throws Exception
{
Class.forName("oracle.jdbc.OracleDriver");
System.out.println("driver is loaded");
con=DriverManager.getConnection("jdbc:oracle:thin:@John-
pc:1521:xe","system","system");
System.out.println("connection is opend");
}
void insertTest()throws Exception
{
PreparedStatement pstmt=con.prepareStatement("insert into student values(?,?,?) ");
Scanner s=new Scanner(System.in);
String Choice="yes";
while(Choice.equals("yes"))
{
System.out.println("enter student id");
int sid=s.nextInt();
System.out.println("enter student name");
String sname=s.next();
System.out.println("enter Student marks");
int marks=s.nextInt();
```

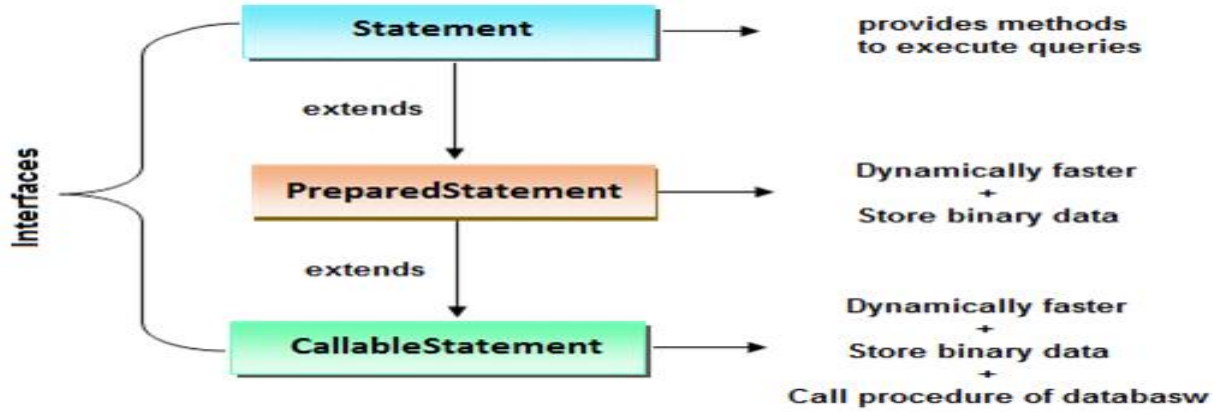
```
//setting the values
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3,marks);
int i=pstmt.executeUpdate();
System.out.println(i+"Row inserted");
System.out.println("do you want to inset another row(Yes/no)");
Choice=s.next();
}//end while
pstmt.close();
}
void closeConnection()throws Exception
{
con.close();
System.out.println("connection is closed");
}
publicstaticvoid main(String[] args) throws Exception
{
PrepardTest1 pt=new PrepardTest1();
pt.openConnection();
pt.insertTest();
pt.closeConnection();
}
}
```

Callablestatement in Jdbc

To call the procedures and functions of a database, CallableStatement interface is used.

CallableStatement is a derived Interface of preparedStatement. It has one additional feature over PreparedStatement that is calling procedures and function of a database.

Because of CallableStatement is inherited from PreparedStatement all the features of PreparedStatement are also available with CallableStatement.



Create object of CallableStatement

The prepareCall() method of Connection interface returns the instance of CallableStatement. To create a reference of CallableStatement we have two syntaxes one with command and the other is wih calling procedure or function.

Syntax of prepareCall() method

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

Syntax

```
CallableStatement cstmt=con.prepareCall("sql command");
```

Syntax

```
CallableStatement cstmt=con.prepareCall("{call procedure name or function name}");
```

Syntax

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

Resultsetmetadata Interface in JDBC

The metadata means data about data, in case of database we get metadata of a table like total number of column, column name, column type etc.

While executing a select operation on a database if the table structure is already known for the programmer, then a programmer of JDBC can read the data from ResutlSet object directly. If the table structure is unknown then a JDBC programmer has to take the help of **ResultSetMetadata**.

A ResultSetMetaData reference stores the metadata of the data selected into a ResutlSet object.

How to get the object of ResultSetMetaData ?

To obtain a object of ResultSetMetaData, we need to call getMetaData() method of ResutlSet object..

Syntax

```
ResultSetMetaData rsmd=rs.getMetaData();
```

Methods of ResultSetMetaData

The following are the methods called on ResultSetMetaData reference.

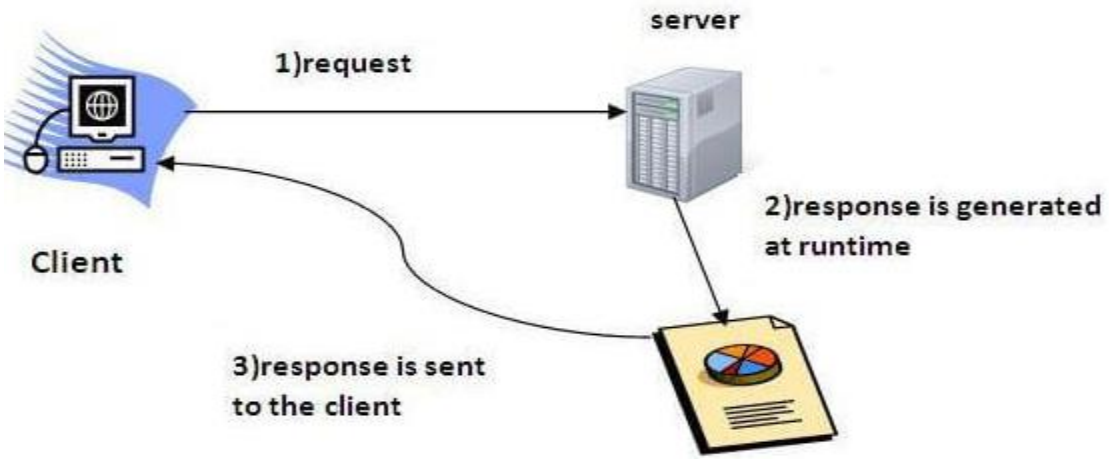
	method	Discription
1	getColumnCount()	To find the number of columns in a ResultSet
2	getColumnName()	To find the column name of a column index.
3	getColumnTypeName()	To find data type of a column.
4	getColumnDisplaySize()	To find size of a column.

CHAPTER 4 SERVLET

INTRODUCTION TO SERVLET

Servlet is a java program, exist and executes in j2ee servers, used to receive the http protocol request, process and send response to client.

Using Servlet, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.



The above figure shows HTTP request and response generated with the help of servlet container in server.

ADVANTAGE OF SERVLET

Better performance: Because it creates a thread for each request not process (like CGI).

Portability: Because it uses java language and java is robust language.

Robust: Servlet are managed by JVM so no need to worry about memory leak, garbage collection etc.

Secure: Because it uses java language and java is a secure language.

Features of Servlet

Servlet is a java program, exist and executes in j2ee servers, used to receive the http protocol request, process and send response to client.

Using Servlets, we can collect input from users through web page forms, present records from a database or another source that may be any servlet, JSP or html page and create web pages dynamically.

Container in Servlet

Container provides runtime environment for Java2ee (j2ee) applications. A web container is a predefined application provided by a server, its takes care of Servlet and JSP.

In console based java applications a class which contains main method acts as a container for other classes.

Container in Servlet Example

```
class Vachile
{
void speed()
{
System.out.println("Speed limit is 40 km/hr");
```

```
}
}
class Mainclass
{
public static void main(String args[])
{
Vachile v=new Vachile();
v.speed();
}
}
```

Output

```
Speed limit is 40 km/hr
```

Here Mainclass is providing run-time support for vehicle class so main class is called a container.

In GUI based applications a frame acts as a container for other **awt** components like button, textfield etc.

Operations of Servlet Container.

- Life Cycle Management
- Communication Support
- Multithreaded support
- Security etc.

1. Life cycle management: Servlet and JSP are dynamic resources of java based web application. The Servlet or JSP will run on a server and at server side. A container will take care

about life and death of a Servlet or JSP.
A container will instantiate, Initialize, Service and destroy of a Servlet or JSP. It means life cycle will be managed by a container.

2. Communication Support: If Servlet or JSP wants to communicate with server than its need some communication logic like socket programming. Designing communication logic is increase the burden on programmers, but container act as a mediator between a server and a Servlet or JSP and provides communication between them.

3. Multithreading: A container creates a thread for each request, maintains the thread and finally destroys it whenever its work is finished.

4. Security: A programmer is not required to write security code in a Servlet/JSP. A container will automatically provide security for a Servlet/JSP.

Servlet Interface

It is an interface to define a Servlet, the implementation class of this Servlet should override all methods of Servlet interface.
Servlet interface needs to be implemented for creating any Servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the Servlet, to service the requests, and to destroy the Servlet and 2 non-life cycle methods.

Methods of Servlet interface

Method	Description
public void init(ServletConfig config)	initializes the Servlet. It is the life cycle method of Servlet and invoked by the web container only once.

public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that Servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about Servlet such as writer, copyright, version etc.

GENERICSERVLET CLASS

GenericServlet class Implements Servlet, ServletConfig and Serializable Interfaces. It provides the implementation of all the methods of these Interfaces except the service method. GenericServlet class can handle any type of request so it is protocol Independent. You may create a generic Servlet by inheriting the GenericServlet class and providing the Implementation of the service method.

This is implemented abstract class for Servlet Interface, have the implementation for all methods of Servlet interface except service method.

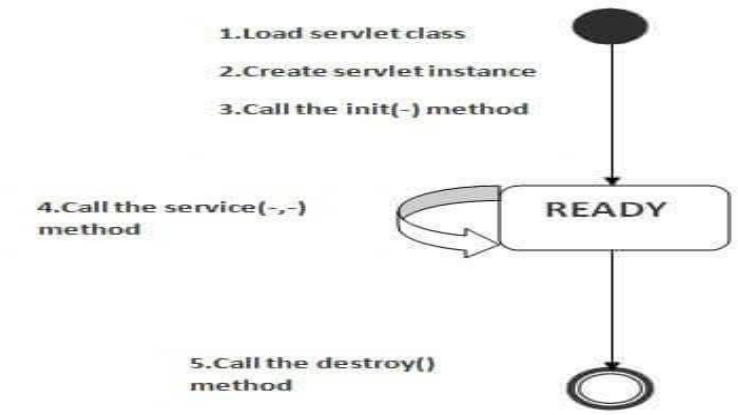
EXAMPLE OF SERVLET BY INHERITING THE GENERICSERVLET CLASS

```
import java.io.*;
import javax.servlet.*;
public class GenericServletDemo extends GenericServlet
{
public void service(ServletRequest req,ServletResponse resp)
```

```
throws IOException,ServletException
{
res.setContentType("text/html");
PrintWriter out=resp.getWriter();
out.print("<html><body>");
out.print("<b>Example of GenericServlet</b>");
out.print("</body></html>");
} }
```

LIFE CYCLE OF SERVLET

The web container maintains the life cycle of a Servlet instance or object.



1. Loading (Servlet class is loaded)
2. Installation (Servlet instance is created)
3. Initialization (init method is invoked)
4. Service providing (service method is invoked)
5. Destroying (destroy method is invoked)

As displayed in the above diagram, there are three states of a Servlet: new, ready and end. The Servlet is in new state if Servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, Servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1. Servlet class is loaded

The classloader is responsible to load the Servlet class. The Servlet class is loaded when the first request for the Servlet is received by the web container.

2. Servlet instance is created

The web container creates the instance of a Servlet after loading the Servlet class. The Servlet instance is created only once in the Servlet life cycle.

3. init method is invoked

The web container calls the init method only once after creating the Servlet instance. The init method is used to initialize the Servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

Syntax

```
public void init(ServletConfig config) throws ServletException
```

4. service method is invoked

The web container calls the service method each time when request for the Servlet is received. If Servlet is not initialized, it follows the first three steps as described above then calls the service method. If Servlet is initialized, it calls the service method. Notice that Servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

Syntax

```
public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException
```

5. Destroy method is invoked

The web container calls the destroy method before removing the Servlet instance from the service. It gives the Servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

Syntax

```
public void destroy()
```

Servlet Life Cycle Example

```
import javax.servlet.*;
import java.io.*;

public class myservlet extends GenericServlet
{
 public void init(ServletConfig sc)
{
  System.out.println("init executed...");
}
public void service(ServletRequest req, ServletResponse resp)throws IOException,
ServletException
{
System.out.println("service executed...");
PrintWriter out=resp.getWriter();
resp.setContentType("text/html");
out.println("plz observe output on server console window");
}
public void destroy()
{
```

```
 System.out.println("Distroy executed...");
}
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>srv</servlet-name>
<servlet-class>myservlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>srv</servlet-name>
<url-pattern>/ms</url-pattern>
</servlet-mapping>
</web-app>
```

Content Type in Servlet

Content Type is also known as **MIME Type**. MIME stand for **Multipurpose internet Mail Extension**. It is a HTTP header that provides the description about what are you sending to the browser (like send image, text, video etc.).

This is the format of http protocol to carry the response contains to the client..

Example: Suppose you send html text based file as a response to the client the MIME type specification is

Syntax

```
response.setcontentType("text/html");
```

MIME type have two parts, They are:

- Base name
- Extension name

Base name: It is the generic name of file.

Extension name: It is extension name for specific file type.

The supporting MIME type by http protocol are:

File	MIME Type	Extension
Xml	text/xml.	.xml.
HTML	text/html.	.html.
Plaintext File	text/plain.	.txt.
PDF	application/pdf.	.pdf.
gif Image	image/gif.	.gif.
JPEG Image	image/jpeg.	.jpeg.
PNG Image	image/x-png.	.png.
MP3 Music File	audio/mpeg.	.mp3.
MS Word Document	application/msword.	.doc.
Excel work sheet	application/vnd.ms-sheet.	.xls.

Power Point Document	application/vnd.ms-powerpoint.	.ppt.

Syntax

```
<web-app>
<servlet>
<servlet-name>alias name</servlet-name>
<servlet-class>fully qualified class name</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>alias name</servlet-name>
<url-pattern>/url pattern</url-pattern>
</servlet-mapping>
</web-app>
```

Welcome File Configuration in Servlet

A **welcome file** is the file that is invoked automatically by the server, if you don't specify any file name. Welcome file is a default starting page of the website.

The welcome-file-list element of web-app, is used to define a list of welcome files. Its sub element is welcome-file that is used to define the welcome file.

By default server looks for the welcome file in following order:

1. index.html
2. index.htm
3. index.jsp

Note: If welcome file is not configure and index.html or index.jsp does not exits in an application then container sends http status 404 message to the browser.

If you have specified welcome-file in web.xml, and all the files index.html, index.htm and index.jsp exists, priority goes to welcome-file.

If welcome-file-list entry doesn't exist in web.xml file, priority goes to index.html file then index.htm and at last index.jsp file.

Let's see the web.xml file that defines the welcome files.

web.xml

Example

```
<web-app>
 ....
   <welcome-file-list>
  <welcome-file>home.html</welcome-file>
  <welcome-file>home.jsp</welcome-file>
 </welcome-file-list>
</web-app>
```

Now, home.html and default.html will be the welcome files.

If you have the welcome file, you can directory invoke the project as given below:

Syntax & Example :

```
http://hostname:portno/contextroot/urlpatternofservlet.
```

```
  http://localhost:8888/myproject/
```

SERVLET FIRST PROGRAM

Servlet programming is very simple but you need some basic knowledge for example interface, abstract class, exception handling, file handling etc.

STEPS TO WRITE SERVLET PROGRAM

THERE ARE GIVEN 6 STEPS TO CREATE A SERVLET EXAMPLE. THESE STEPS ARE REQUIRED FOR ALL THE SERVERS. THE SERVLET EXAMPLE CAN BE CREATED BY THREE WAYS:

1.BY IMPLEMENTING SERVLET INTERFACE

2.BY INHERITING GENERICSERVLET CLASS, (OR)

3.BY INHERITING HTTPSERVLET CLASS

THE MOSTLY USED APPROACH IS BY EXTENDING HTTPSERVLET BECAUSE IT PROVIDES HTTP REQUEST SPECIFIC METHOD SUCH AS DOGET(), DOPOST().HERE, WE ARE GOING TO USE APACHE TOMCAT SERVER IN THIS EXAMPLE. THE STEPS ARE AS FOLLOWS:

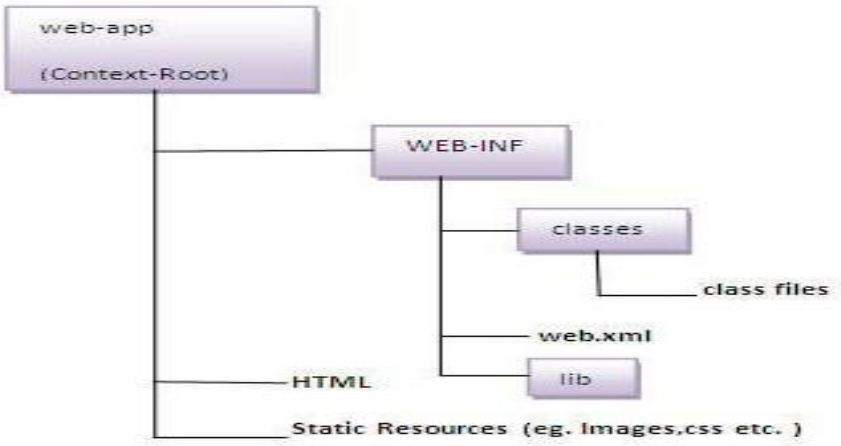
1.CREATE A DIRECTORY STRUCTURE

2.CREATE A SERVLET
3.COMPILE THE SERVLET
4.CREATE A DEPLOYMENT DESCRIPTOR
5.START THE SERVER AND DEPLOY THE PROJECT
6.ACCESS THE SERVLET

1. Create a directory structure

The **directory structure** defines that where to put the different types of files so that web container may get the information and responds to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. see the directory structure that must be followed to create the servlet.



2. **Create a Servlet**

Three ways to do this:- By implementing the Servlet interface

By inheriting the GenericServlet class

By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle

http requests such as doGet(), doPost, doHead() etc.In this example we are going to create a servlet that extends the HttpServlet class.

In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. This is default method.

3.Compile a Servlet

For compiling the Servlet, jar file is required to be loaded. & servlet-api.jar file should be attached with Apache Tomcat server.

Use this path to add external jar:- Go to project ➔Right click on project ➔ Select Build path ➔ Configure build path ➔ Select Library ➔ Select Add External jars ➔ Brows your servlet-api.jar file from predefined location ➔ Apply and Close.

4. Create Deployment Descriptor(web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets

the information about the servet to be invoked.

There are many elements in the web.xml file. The following structure shows contents in .xml file.

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Where **<web-app>** represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5. **Start the server & deploy the project**

   One Time Configuration for Apache Tomcat Server

   You need to perform 2 tasks:

   set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).

   Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

   Apache tomcat that needs to install

Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

How to deploy the servlet project?

Copy the project and paste it in the webapps folder under apache tomcat.

6. **How to access the servlet**

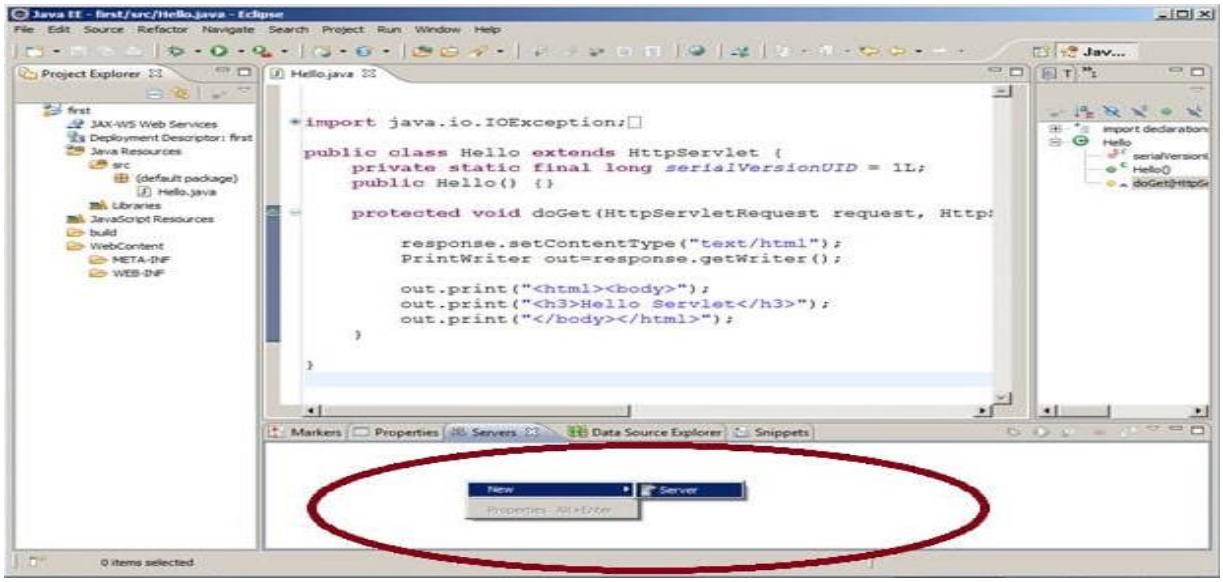
   http://hostname:portno/contextroot/urlpatternofservlet.

   For example:

   http://localhost:9999/demo/welcome

How to configure tomcat server in Eclipse for the first time ?

For configuring the tomcat server in eclipse IDE, **click on servers tab at the bottom side of the IDE -> right click on blank area -> New -> Servers -> choose tomcat then its version -> next -> click on Browse button -> select the apache tomcat root folder previous to bin -> next -> addAll -> Finish.**



CREATING SERVLET EXAMPLE IN ECLIPSE IDE

1) Create the dynamic web project:

For creating a dynamic web project **click on File Menu -> New ->**

Project..-> Web -> dynamic web project -> write your project name e.g. first -> Finish.

2) Create the servlet in eclipse IDE:

For creating a servlet, **explore the project by clicking the + icon**

-> explore the Java Resources -> right click on src -> New ->

servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish.

3) add jar file in eclipse IDE:

For adding a jar file, **right click on your project -> Build Path ->**

Configure Build Path -> click on Libraries tab in Java Build Path -> click on Add External JARs button -> select the servlet-api.jar file under tomcat/lib -> ok.

4) Start the server and deploy the project:

For starting the server and deploying the project in one step, **Right**

click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.

Now tomcat server has been started and project is deployed. To access the servlet write the url pattern name in the URL bar of the browser.

WRITE A WEB APPLICATON TO SEND HELLO WORD AS RESPONSE TO CLIENT USING SERVLET.

FIRSTSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;


public class FirstServlet extends HttpServlet {
  public void service(HttpServletRequest request, HttpServletResponse response)throws
IOException, ServletException
 {
  // get request parameter
  // business operation
  String resultvalue="<body bgcolor="cyan" text="red"> <h1> hello word</h1></body>";
  // prepare response
```

```
  resp.setContentType("text/html");
  printWriter out=resp.getWriter();
  // send response
  out.print(resultvalue);
  out.close();
  }
}
```

WEB.XML

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
</web-app>
```

Interfaces used in servlet API

1. Servlet Request Interface

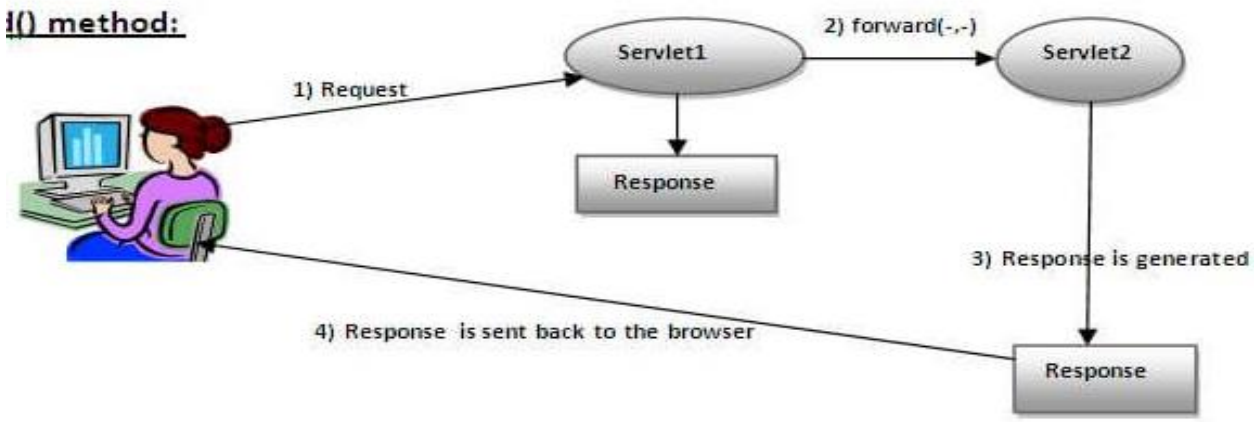
- An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc
- Example of ServletRequest to display the name of the user

- In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

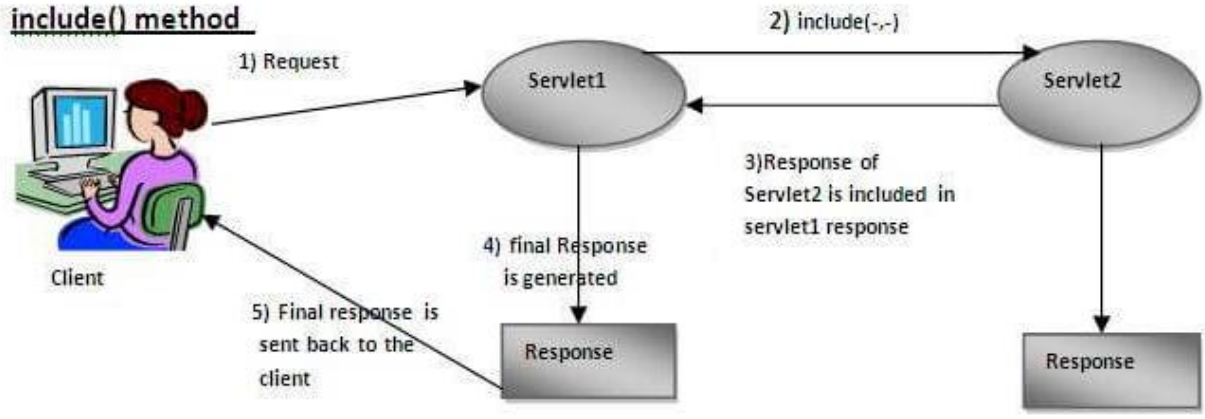
2. Requser Dispatcher Interface

- The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- This interface can also be used to include the content of another resource also.There are two methods defined in the RequestDispatcher interface.
- **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

The following figure shows forward method:



The following figure shows include method:



Example:

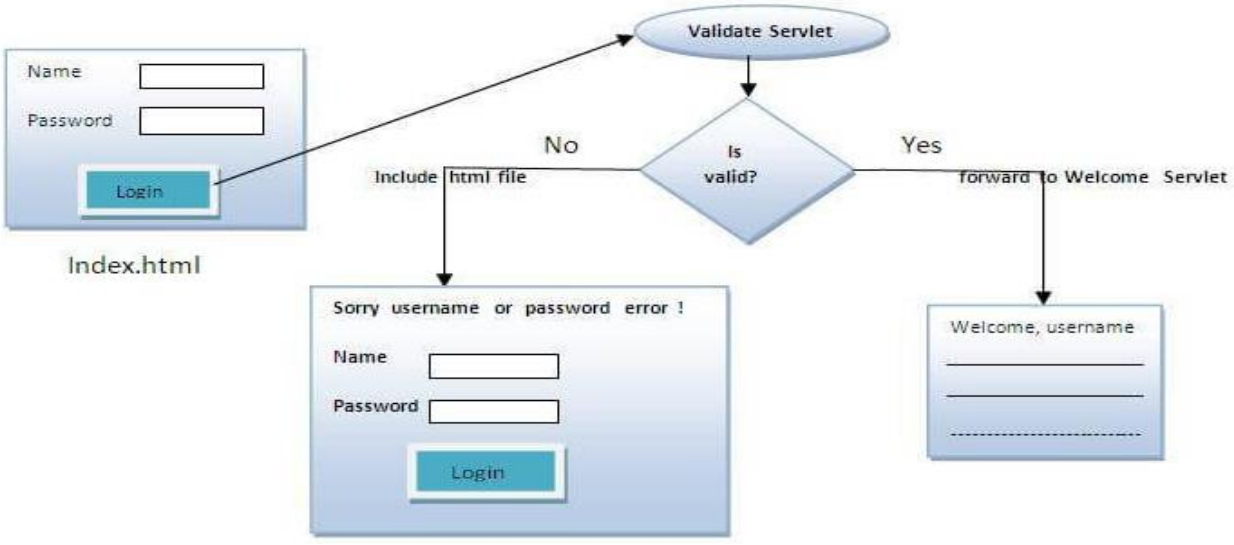
In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message.

index.html file: for getting input from the user.

Login.java file: a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.

WelcomeServlet.java file: a servlet class for displaying the welcome message.

web.xml file: a deployment descriptor file that contains the information about the servlet.



3. SendRedirect Interface

The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
It accepts relative as well as absolute URL.
It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.
In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server.
Difference between forward and sendRedirect method:

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

4. ServletConfig Interface in Servlet

An object of ServletConfig is created by the web container for each servlet using its initialization phase. This object can be used to get configuration information from web.xml file.

An object of ServletConfig is available to the Servlet during its execution, once the servlet execution is completed, automatically ServletConfig interface object will be removed by the container.

When ever compiler executes init() method then the ServletConfig will be created in general. An object of ServletConfig contain the

An object of ServletConfig contain the <init-param> data in the form of key,value pairs, here the keys represents init param names and values are its values, which are represented in the web.xml file

Advantage of ServletConfig

If the configuration information is modified from the web.xml file, we don't need to change the Servlet. So it is easier to manage the web application if any specific content is modified from time to time.

The core advantage of ServletConfig is that you don't need to edit the Servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

- **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
- **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
- **public String getServletName():**Returns the name of the Servlet.
- **public ServletContext getServletContext():**Returns an object of ServletContext.

How to Get ServletConfig Object into Servelt

An object of ServletConfig can be obtained in 2 ways,

Way 1. **Syntax**

```
ServletConfig conf = getServletConfig();
```

In the above statement, we are directly calling getServletConfig() method as it is available in Servlet interface, inherited into GenericServlet and defined and further inherited into HttpServlet and later inherited into our own servlet class.

Way 2. **Syntax**

```
ServletConfig object will be available in init() method of the servlet.
  public void init(ServletConfig config)
{
  // ......
}
```

So finally we are able to create ServletConfig object in our servlet class, then how to get the data from that object ?

How to Retrieve Data from ServletConfig Interface Object

In order to retrieve the data of the ServletConfig we have two methods, which are present in ServletConfig interface.

Syntax

```
public String getInitParameter("param name");
public Enumeration getInitParameterNames();
```

SERVLETCONTEXT INTERFACE IN SERVLET

ServletContext is one of pre-defined interface available in javax.servlet.*; Object of ServletContext interface is available one per web application. An object of ServletContext is automatically created by the container when the web application is deployed.

Assume there exist a web application with 2 servlet classes, and they need to get some technical values from web.xml, in this case ServletContext concept will works great, i mean all servlets in the current web application can access these context values from the web.xml but its not the case in ServletConfig, there only particular servelet can access the values from the web.xml which were written under <servlet> tag, hope you remember. Have doubt ? just check Example of ServletConfig.

HOW TO GET SERVLETCONTEXT OBJECT INTO OUR SERVLET CLASS

In servlet programming we have 3 approaches for obtaining an object of ServletContext interface

Way 1.

SYNTAX

```
ServletConfig conf = getServletConfig();
ServletContext context = conf.getServletContext();
```

First obtain an object of ServletConfig interface ServletConfig interface contain direct method to get Context object, getServletContext();.

Way 2.

Direct approach, just call getServletContext() method available in GenericServlet [pre-defined]. In general we are extending our class with HttpServlet, but we know HttpServlet is the sub class of GenericServlet.

SYNTAX

```
public class Java4s extends HttpServlet
{
public void doGet/doPost(-,-)
{
//
}
ServletContext ctx = getServletContext();
}
```

Way 3.

We can get the object of ServletContext by making use of HttpServletRequest object, we have direct method in HttpServletRequest interface.

SYNTAX

```
public class Java4s extends HttpServlet
{
public void doGet/doPost(HttpServletRequest req,-)
{
ServletContext ctx = req.getServletContext();
}
}
```

HOW TO RETRIEVE DATA FROM SERVLETCONFIG INTERFACE OBJECT

ServletContext provide these 2 methods, In order to retrieve the data from the web..xml [In web.xml we have write <context-param> tag to provide the values, and this <context-param> should write outside of <servlet> tag as context should be accessed by all servlet classes].

In general database related properties will be written in this type of situation, where every servlet should access the same data.

SYNTAX

```
public String getInitParameter("param name");
public Enumeration getInitParameterNames();
```

Difference Between dopost and doget in Servlet

Http protocol mostly use either get or post methods to transfer the request. post method are generally used whenever you want to transfer secure data like password, bank account etc.

Sr. No	Get method	Post method

1	Get Request sends the request parameter as query string appended at the end of the request.	Post request send the request parameters as part of the http request body.
2	Get method is visible to every one (It will be displayed in the address bar of browser).	Post method variables are not displayed in the URL.
3	Restriction on form data, only ASCII characters allowed.	No Restriction on form data, Binary data is also allowed.
4	Get methods have maximum size is 2000 character.	Post methods have maximum size is 8 mb.
5	Restriction on form length, So URL length is restricted	No restriction on form data.
6	Remain in browser history.	Never remain the browser history.

EXCEPTION HANDLING IN SERVLET

The process of converting system error messages into user friendly error message is known as **Exception handling**. This is one of the powerful feature of Java to handle run time error and maintain normal flow of java application.

EXCEPTION

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

- Programetically Exception Handling mechanism
- Declarative Exception Handling mechanism

PROGRAMETICALLY EXCEPTION HANDLING MECHANISM

The approach to use try, catch block in java code to handle exceptions is known as Programetically Exception Handling mechanism.

INDEX.HTML

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

EXCEPTION HANDLING IN SERVLET

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
 public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String n=request.getParameter("userName");
  out.print("Welcome "+n);
  Cookie ck=new Cookie("uname",n);//creating cookie object
  response.addCookie(ck);//adding cookie in the response

  //creating submit button
  out.print("<form action='servlet2'>");
  out.print("<input type='submit' value='continue'>");
  out.print("</form>");
   out.close();
```

```
        }catch(Exception e){System.out.println(e);}
    }
}
```

DECLARATIVE EXCEPTION HANDLING MECHANISM

The approach to use xml tags in web.xml file to handle the exception is known as declarative exception handling mechanism.

This mechanism is usful if exception are common for mote than one servlet program. In real time application this mechanism is widely use.

ERROR.HTML

```
<html>
<body>
<p> Oooops..... page not found</p>
</body>
</html>
```

MYSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;


public class FirstServlet extends HttpServlet {


  public void service(HttpServletRequest request, HttpServletResponse response)throws
IOException, ServletException
  {
```

```
  // get request parameter
  // business operation
  String resultvalue="<body bgcolor="cyan" text="red"> <h1> hello word</h1></body>";
  // prepare response
  resp.setContentType("text/html");
  printWriter out=resp.getWriter();

  // send response
  out.print(resultvalue);
  out.close();
  }
}
```

WEB.XML

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>Myservlet</servlet-class>
</servlet>
<error-page>
<exception-type>java.lang.NumberFormateException</exception-type>
<location>/error.html</location>
</error-page>
</web-app>
```

WAR FILE IN SERVLET

A war (web archive) File is a compressed format of files of a web project. It may have servlet, xml, jsp, image, html, css, js etc.

ADVANTAGE OF WAR FILE

Saves time: The war file combines all the files into a single unit. It is a compressed format of all files, So it takes less time while transferring file from client to server.

HOW TO CREATE WAR FILE?

To create war file, we need jar tool of JDK. We need to use -c switch of jar, to create the war file.

Go inside the project directory of your project (outside the WEB-INF), then write the following command:

SYNTAX

```
jar -cvf projectname.war *
```

Here, -c is used to create file, -v to generate the verbose output and -f to specify the archive file name.

The * (asterisk) symbol signifies that all the files of this directory (including sub directory).

HOW TO EXTRACT WAR FILE MANUALLY?

To extract the war file, you need to use -x switch of jar tool of JDK. Let's see the command to extract the war file.

SYNTAX

```
jar -xvf projectname.war
```

SESSION TRACKING IN SERVLET

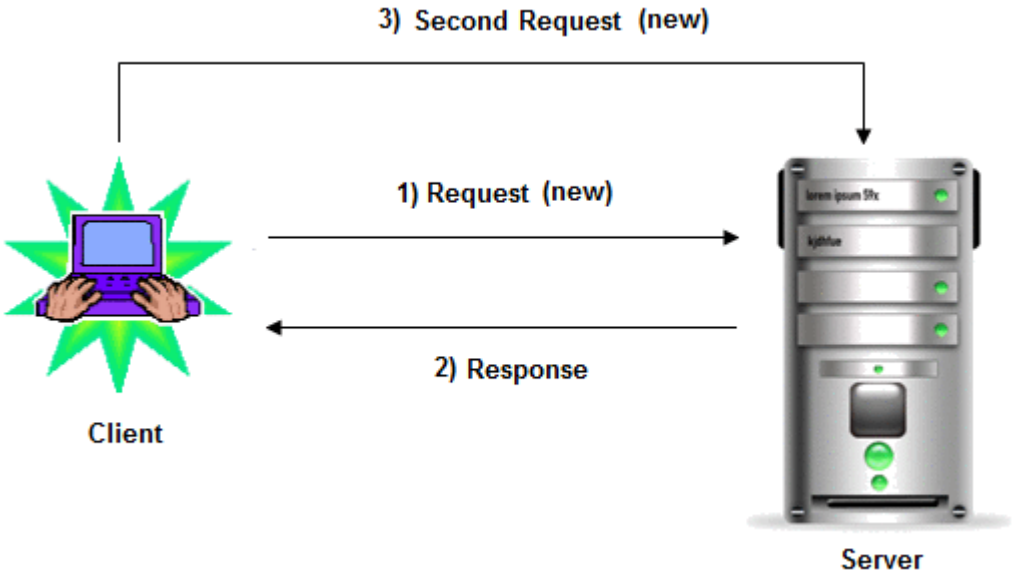
Session is the conversion of user within span of time. In general meaning particular interval of time.**Tracking** is the recording of the thing under session.

Session Tracking is remembering and recording of client conversion in span of time. It is also called as session management.

If web application is capable of remembering and recording of client conversion in span of time then that web application is called as **stateful web application**.

WHY DO WE NEED SESSION TRACKING ?

- Http protocol is stateless, to make stateful between client and server we need Session Tracking.
- Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.
- Http protocol is stateless, that means each request is considered as the new request. You can see in below image.



WHY TO USE SESSION TRACKING ? :

To recognize the user It is used to recognize the particular user.

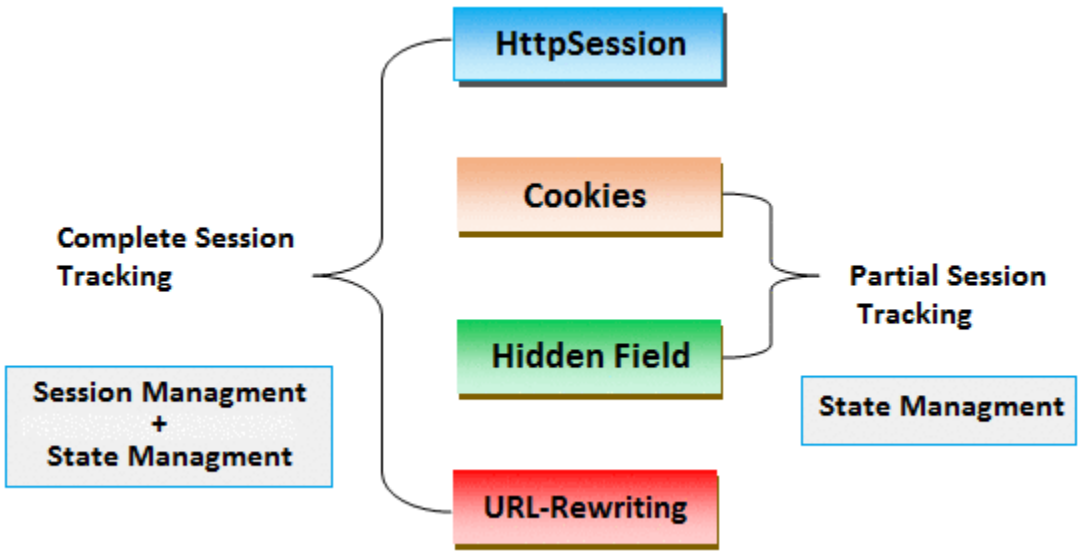
WHY HTTP IS DESIGN AS STATELESS PROTOCOL ?

If Http is stateful protocol for multiple requests given by client to web application single connection will be used between browser and web server across the multiple requests. This may make clients to engage connection with web server for long time event though the connection are ideal. Due to this the web server reach to maximum connections even though most of its connection are idle. To overcome this problem Http is given as stateless.

SESSION TRACKING TECHNIQUES

Servlet technology allows four technique to track conversion, they are;

- Cookies
- URL Rewriting
- Hidden Form Field
- HttpSession



COOKIES HANDLING IN SERVLET

Cookies are text files stored on the client computer and they are kept for various information like name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookies are created using Cookie class present in Servlet API. Cookies are added to response object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used access the cookies that are added to response object.

In Http Session technique, container internally generates a cookies for transferring the session ID between server and client. Apart from container generated cookie a servlet programmer can also generate cookies for storing the data for a client.

HOW COOKIE WORKS

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser (chrome, firefox) at client side. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

WHEN USE COOKIES ?

When session ID is not required and when less number of input values are submitted by client in that case in place of using HttpSession Technique you can use cookies Technique to reduce the burden on server.

POINTS TO REMEMBER

- Cookies is pressistance resource which is stores at client location.
- We can store 3000 cookies in cookies file at a time.
- The cookies are introduced by net scape communication.
- Cookies files exist up to 3 year.
- Size of cookies is 4 kb.

TYPE OF COOCKIES

There are two types of cookies, those are given below;

- In-memory cookies or pre session cookies
- Persistent cookies

In-memory cookies: By default cookie is in-memory coockie, This type of cookie is lives until that browser is destroy(close). It is valid for **single session** only. It is removed each time when user closes the browser.

Persistent cookies: Presestent cookie lives on a browser until its expiration time is reached it means , eventhough you close or reopen the browser but still the cookie exists on the browser. It is valid for **multiple session**. It is not removed each time when user closes the browser. It is removed only if user logout or signout.

COOKIE CLASS

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a some constructor and methods for cookies.

CONSTRUCTOR OF COOKIE CLASS

Constructor	Description
Cookie()	Used for constructs a cookie.
Cookie(String name, String value)	Used for constructs a cookie with a specified name and value.

METHODS OF COOKIE CLASS

Methods	Description
public void setMaxAge(int expiry)	It is used for Sets the maximum age of the cookie in seconds.

public String getName()	It is used for Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	It is used for Returns the value of the cookie.
public void setName(String name)	It is used for changes the name of the cookie.
public void setValue(String value)	It is used for changes the value of the cookie.
public void addCookie(Cookie ck)	It is method of HttpServletResponse interface which is used to add cookie in response object.
public Cookie[] getCookies()	It is method of HttpServletRequest interface which is used to return all the cookies from the browser.

CREATE COOKIES

To create cookies you need to use Cookie class of javax.servlet.http package.

SYNTAX

```
Cookie c=new Cookie(name, value);  // here name and value are string type
```

ADD COOKIES

To add a cookie to the response object, we use addCookie() mehtod.

SYNTAX

```
Cookie c=new Cookie();     //creating cookie object
response.addCookie(c1); //adding cookie in the response
```

READ COOKIES FOR BROWSER

To read Cookies from browser to a servlet, we need to call getCookies methods given by request object and it returns an array type of cookie class.

SYNTAX

```
response.addCookie(c1);
Cookie c[]=request.getCookie();
```

EXAMPLE OF SESSION TRACKING BY USING COOKIES

INDEX.HTML

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FIRSTSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{

  response.setContentType("text/html");
```

```
   PrintWriter out = response.getWriter();
   String n=request.getParameter("userName");
   out.print("Welcome "+n);

   Cookie ck=new Cookie("uname",n);//creating cookie object
   response.addCookie(ck);//adding cookie in the response

   //creating submit button
   out.print("<form action='servlet2'>");
   out.print("<input type='submit' value='continue'>");
   out.print("</form>");
  out.close();
      }catch(Exception e){System.out.println(e);}
 }
}
```

SECONDSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
```

```
    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());
   out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

HIDDEN FORM FIELD

Tracking client conversion using Html hidden variables in secure manner is known as hidden form field.

HOW TO USE HIDDEN FORM FIELD ?

In Hidden Form Field we are use html tag is <input type="hidden"> and with this we assign session ID value.

SYNTAX

```
<input type="hidden" name="uname" value="porter">
```

EXAMPLE OF SESSION TRACKING BY USING HIDDEN FORM FIELD

INDEX.HTML

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FIRSTSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      //creating form that have invisible textfield
      out.print("<form action='servlet2'>");
      out.print("<input type='hidden' name='uname' value='"+n+"'>");
      out.print("<input type='submit' value='continue'>");
      out.print("</form>");
      out.close();

        }
                  catch(Exception e){System.out.println(e);}
   }
}
```

SECONDSERVLET.JAVA

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    //Getting the value from the hidden field
    String n=request.getParameter("uname");
    out.print("Hello "+n);

    out.close();
    }
                catch(Exception e){System.out.println(e);}
  }
}
```

URL REWRITING IN SERVLET

URL Rewriting track the conversion in server based on unique session ID value.

WHEN USE URL REWRITING ?

If the client has disabled cookie in the browser then coockie are not work for session management. In that case you can use URL rewriting technique for session managment. URL rewriting will always work.

EXAMPLE OF SESSION TRACKING BY USING URL REWRITING

INDEX.HTML

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FIRSTSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{
        response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);
     HttpSession session=request.getSession();
      session.setAttribute("uname",n);
    out.print("<a href='servlet2'>visit</a>");
    out.close();
            }catch(Exception e){System.out.println(e);}
   }
}
```

SECONDSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //getting value from the query string
      String n=request.getParameter("uname");
      out.print("Hello "+n);

      out.close();

        }
          catch(Exception e){System.out.println(e);}
   }
}
```

HTTPSESSION IN SERVLET

HttpSession is another kind of session management technique, In this technique create a session object at server side for each client.

Session is available until the session time out, until the client log out. The default session time is 30 minutes and can configure explicit session time in web.xml file.

CONFIGURE SESSION TIME IN WEB.XML

EXAMPLE

```
<web-app>
<session-config>
<session-timeout>40</session-timeout>
</session-config>
</web-app>
```

HTTPSESSION API

Http session is an interface define in java.http package.

GETTING SESSION OBJECT

EXAMPLE

```
HttpSession hs=req.getSession();  // create new session object
```

METHODS OF HTTPSESSION INTERFACE

Method	Description
public HttpSession getSession():	It returns the current session associated with this request, or if the request does not have a session, creates one.
public HttpSession getSession(boolean create)	It returns the current HttpSession associated with this request or, if there is no current

	session and create is true, returns a new session.
public String getId()	It returns a string containing the unique identifier value.
public long getCreationTime()	It returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public long getLastAccessedTime()	It returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
public void invalidate()	Invalidates this session then unbinds any objects bound to it.

EXAMPLE OF SESSION TRACKING BY USING HTTPSESSION

INDEX.HTML

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FIRSTSERVLET.JAVA

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      HttpSession session=request.getSession();
      session.setAttribute("uname",n);
     out.print("<a href='servlet2'>visit</a>");
     out.close();
            }catch(Exception e){System.out.println(e);}
   }
}
```

SECONDSERVLET.JAVA

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class SecondServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      HttpSession session=request.getSession(false);
      String n=(String)session.getAttribute("uname");
      out.print("Hello "+n);

      out.close();
           }catch(Exception e){System.out.println(e);}
   }
}
```

CHAPTER 5 JSP

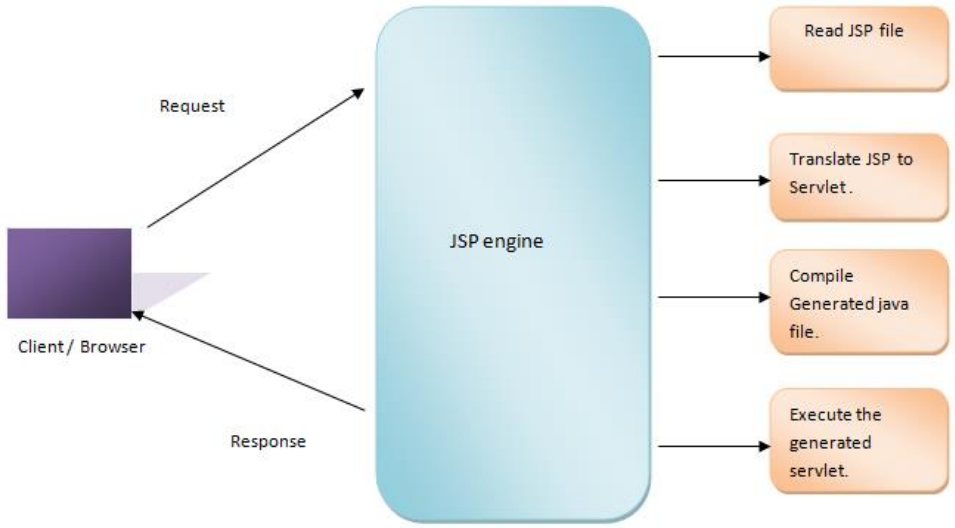
INTRODUCTION TO JSP

JSP technology is used to create dynamic web application same like Servlet technology. It is another web technology given by Sun MicroSystem for the development of dynamic web pages on the client browser. It provides a tag based approach to develop java web components.

JSP have **.jsp** extension, we can directly access these JSP pages from client system browser window. Because jsp pages are contains outside of the WEB-INF folder.

A JSP page consists of Html tags and JSP tags. The jsp pages are easier to maintain than servlet. It provides some additional features such as Expression Language, Custom Tag etc.

A JSP is called as page but not program because a JSP contains totally tags. Every JSP is internally converted into a Servlet by the server container.



WHY JSP ?

The first technology given for the development of web application is CGI. In CGI have some drawback, So Sun MicroSystem develop a new technology is servlet. But working with Servlet Sun MicroSystem identify some problem, Servlet technology need in depth java code and Servlet is failed to attract the programmer.

To overcome the problem with Servlet technology we use jsp technology.

JSP TAG

A JSP page contains both html tags and JSP tags. Html tags will produce static output and JSP tags will produced dynamic output. Because of JSP tags we called as JSP is dynamic web page.

DIFFERENCE BETWEEN JSP AND HTML

Html is a Client side Technology and JSP is a Server side Technology. Some other differences are given below;

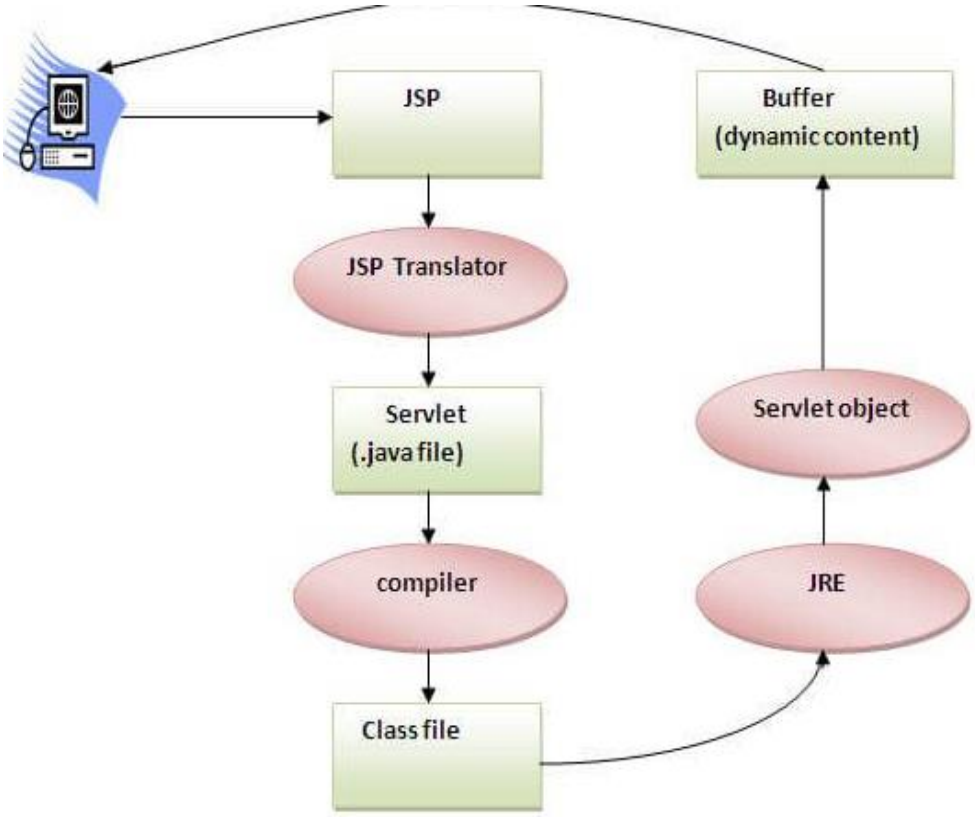
	HTML	JSP
1	Html is given by w3c (World Wide Web Consortium).	JSP is given by SunMicro System.
2	Html generated static web pages.	JSP generated dynamic web pages.
3	It do not allow to place java code inside Html pages.	JSP allows to place java code inside JSP pages.
4	It is Client side technology	It is a Server side technology.
5	Need Html Interpreter to execute these code.	Need JSP container to execute jsp code.
6	It does not allow to place custom tag or third party tag.	It allow to place custom tag or third party tag.

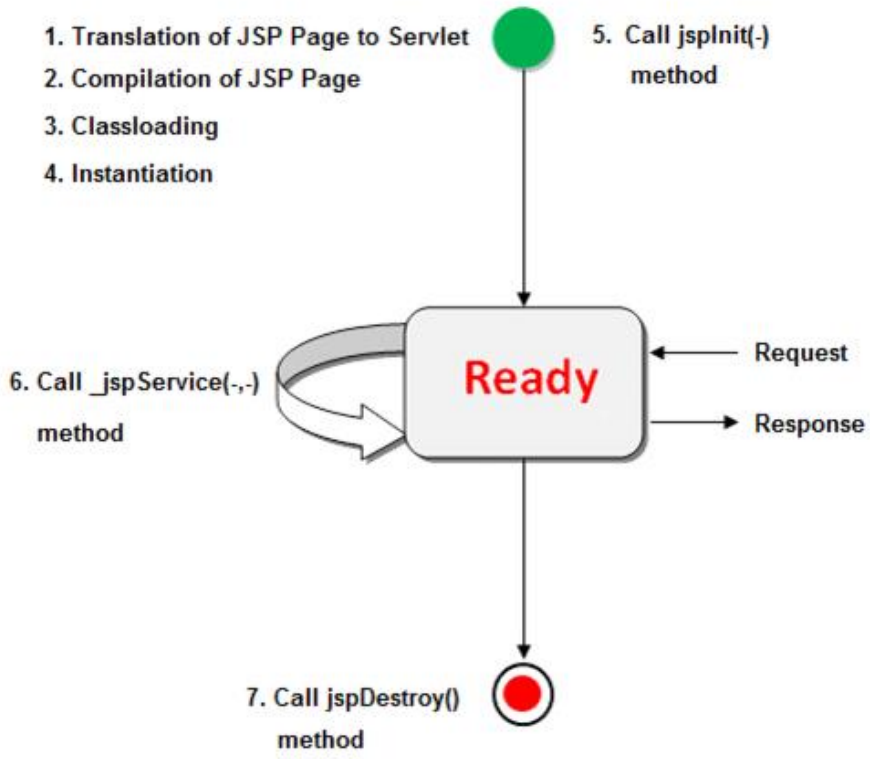
JSP LIFE CYCLE

A **JSP life cycle** can be defined as the entire process from its creation till the destruction which is similar to a Servlet life cycle with an additional step which is required to translate a JSP into Servlet.

LIFE CYCLE OF A JSP PAGE

- Translation of JSP Page to Servlet
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).


JSP
Buffer
(dynamic content)
JSP Translator
Servlet
(.java file)
Servlet object
compiler
JRE
Class file



LIFE CYCLE METHOD OF JSP

- jspInit()
- _jspService()
- jspDestroy()

We can override jspInti(), jspDestroy(). But we can not override _jspService() method.

To inform the programmer that you should not override the service() method the method name is started with '_' symbol.

CONFIGURING JSP FILE

Configuring a JSP into a web.xml file is optional because JSP is a public file to the web application.

A JSP called a public file and servlet is called a private file of the web application. Because JSP files stored in root directory of the web application and Servlet class file stored in sub directory of the web application. Because JSP is the public file, we can directly send a request from a browser by using its file name.

If we want ot configure a JSP in web.xml file the the xml elements are same as Servlet-configuration. We need to replace <servlet-class> element with <jsp-file>

Syntax

```
<web-app>
<servlet>
<servlet-name>test</servlet-name>
<jsp-file>/One</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>test</>
<url-pattern>/srv1</url-pattern>
</servlet-mapping>
</web-app>
```

Note: If we configure a jsp in web.xml then we can send the request to the jsp either by using jsp filename or by using its url-pattern.

Example

```
httpp://localhost:2014/root/One.jsp

                or

httpp://localhost:2014/root/srv1
```

Difference Between Servlet And JSP

	Servlet	JSP
1	Servlet is faster than jsp	JSP is slower than Servlet because it first translate into java code then compile.
2	In Servlet, if we modify the code then we need recompilation, reloading, restarting the server> It means it is time consuming process.	In JSP, if we do any modifications then just we need to click on refresh button and recompilation, reloading, restart the server is not required.
3	Servlet is a java code.	JSP is tag based approach.
4	In Servlet, there is no such method for running JavaScript at client side.	In JSP, we can use the client side validations using running the JavaScript at client side.
5	To run a Servlet you have to make an entry of Servlet mapping into the deployment descriptor file i.e. web.xml file externally.	For running a JSP there is no need to make an entry of Servlet mapping into the web.xml file externally, you may or not make an entry for JSP file as welcome file list.
6	Coding of Servlet is harden than jsp.	Coding of jsp is easier than Servlet because it is tag based.
7	In MVC pattern, Servlet plays a controller role.	In MVC pattern, JSP is used for showing output data i.e. in MVC it is a view.
8	Servlet accept all protocol request.	JSP will accept only http protocol request.
9	In Servlet, aervice() method need to override.	In JSP no need to override service() method.
10	In Servlet, by default session management is not enabled we need to enable explicitly.	In JSP, session management is automatically enabled.
11	In Servlet we do not have implicit object. It means if we want to use an object then we need to get object explicitly form the servlet.	In JSP, we have implicit object support.

12	In Servlet, we need to implement business logic, presentation logic combined.	In JSP, we can separate the business logic from the presentation logic by uses javaBean technology.
13	In Servlet, all package must be imported on top of the servlet.	In JSP, package imported anywhere top, middle and bottom.

Jsp First program

Save and Compile jsp program

JSP program must be save with the **.jsp** extension. And for compile jsp code simply follow step of compilation servlet code.

Example

```
<html>
<body>
<% out.print("Hello word"); %>
</body>
</html>
```

How to run jsp program

After write your jsp code, deploy jsp program in your root directory and start server. Follow below steps to run jsp code.

- Deploy jsp code in root directory
- Start Server
- Visit on browser or give request with url like below

Syntax

```
http://localhost:portnumber/RootDirectory/jspfile
```

Example

http://localhost:2015/jspapplication/index.jsp

Scripting Element

In JSP, java code can be written inside the jsp page using the scriptlet tag. JSP Scripting element are written inside **<% %>** tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page.

Jsp scripting elements are classified into two types are;

- Language Based Scripting Element
- Advance Scripting Elements (Expression Language)

Language Based Scripting Element

These are used to defined script in jsp page, this is traditional approach to define the script in jsp page.

Language Based Scripting Element are classified into 4 types, they are;

- Comment Tag
- Declaration Tag
- Expression Tag
- Scriptlet Tag

Scripting Element Detail

	Start tag	Purpose	End tag	Inside class scope	Inside JSP Scope	Session
JSP Declaration	<% !	Declaration variable and method	% >	yes	no	yes
JSP Expression	< % =	Display response on browser	%>	no	yes	no
JSP Scriptlet	< %	Defining java code	%>	no	no	yes
JSP Comment	< % - -	Comment discription	- - %>	yes	yes	not applicable

JSP Comment

This is used to define comment description in jsp page. In jsp page we can insert three types of comments they are;

- Jsp comment
- Html comment
- Java comment

JSP Comment

This type of comment is called as hidden comment, because it is invisible when a jsp is translated into a servlet internally.

Syntax

```
<%-- Comment discription %>
```

Html Comment

Html comment tag is common for Html, xml and jsp. In a jsp page if we write html comment these comment are visible in the _jspService(-,-) of the internal Servlet.

Syntax

```
<!-- Comment discription -->
```

Java Comment

We can write a single or multiline comment of java in a scriptlet tag.

Java comment are allowed only inside scriptlet tags because we can insert java code in a jsp only at a scriptlet tags.

Syntax

```
// Single line comment
/*
Multiline comment
*/
```

JSP DECLARATION TAG

This is used to declare variable and methods in jsp page will be translate and define as class scope declaration in .java file.

The variable and methods will become global to that jsp. It means in that jsp we can use those variables any where and we can call those method any where.

At the time of translation container inserts the declaration tag into the class. So the variables become instants variables and methods will become instants methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto-generated Servlet, so it does not get memory at each request.

SYNTAX

```
<%! variable declaration or method declaration %>
```

JSP EXPRESSION TAG

Expression tag is used, to find the result of the given expression and send that result back to the client browser. In other words; These are used to show or express the result or response on browser. We can use this as a display result of variable and invoking method.

Each Expression tag of jsp will be internally converted into **out.print()** statement. In jsp out is an implicit object.

One expression tag should contains one java expression only. In a jsp we can use expression tag for any number of times.

SYNTAX

```
<%= expression %>
```

Expression does not need any semicolon (;) by default it places under jsp service() method scope.

Note: In jsp, the implicit object are allowed into the tags, which are inserted into _jspService(-,-). An expression tag goes to _jspService(-,-) only. So implicit object are allowed in the expression tag.

JSP SCRIPTLET TAG

It use to define or insert java code in a jsp. Every java statement should followed by semicolon (;). It will be place into jspService() method.

In a jsp implicit object are allowed in expression tags and scriptlet tags but not in the declaration tags.

SYNTAX

```
<% java code %>
```

JSP Implicit Objects

JSP provides standard or predefined implicit objects, which can use directly in JSP page using **JSP Scriptlet**. The implicit objects are Servlet API class type and created by JSP containers

There are **9 jsp implicit objects**. These objects are created by the web container (JSP containers) that are available to all the jsp pages.

The available implicit objects are out, request, response, page, pageContext, exception, config, session and application.

LIST OF ALL 9 IMPLICIT OBJECT ARE;

	Class type	Object
1	HttpServletRequest	request
2	HttpServletResponse	response
3	ServletConfig	config
4	ServletContex	application

5	HttpSession	session
6	JspWriter/PrintWriter	out
7	Exception	exception
8	PageContext	pagecontext
9	Object	page

All implicit objects of jsp are accessible with in the expression and scriptlet of the jsp, but not accessible in the declaration tags of the jsp.

In a jsp an exception object is available, if isErrorPage is equal to true in page directive.

REQUEST IMPLICIT OBJECT

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Example of request implicit object where we are printing the name of the user with welcome message.

EXAMPLE OF JSP REQUEST IMPLICIT OBJECT

INDEX.HTML

EXAMPLE

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
```

```
</form>
```

welcome.jsp

EXAMPLE

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

RESPONSE IMPLICIT OBJECT

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

EXAMPLE OF RESPONSE IMPLICIT OBJECT : INDEX.HTML

EXAMPLE

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

EXAMPLE

```
<%
response.sendRedirect("http://www.google.com");
%>
```

CONFIG IMPLICIT OBJECT

config object is an implicit object of type ServletConfig and it is created by the container, whenever servlet object is created. This object can be used to get initialization parameter for a particular JSP page.

config object is created by the web container for every jsp page. It means if a web application has three jsp pages then three config object are created.

config object is accessible, only if the request is given to the jsp by using its url pattern, but not with name of the jsp.

EXAMPLE OF CONFIG IMPLICIT OBJECT

INDEX.HTML

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

WEB.XML FILE

```
<web-app>
<servlet>
<servlet-name>Home</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
```

```
<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>
<servlet-mapping>
<servlet-name>Home</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

WELCOME.JSP

```
<%
out.print("Welcome "+request.getParameter("uname"));
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

PAGE IMPLICIT OBJECT

In JSP, page is an implicit object of type Object class. When a jsp is translated to an internal Servlet, we can find the following statement in the service() method ofservlet. Object page=this;

For using this object it must be cast to Servlet type.For example:

EXAMPLE

```
<% (HttpServlet)page.log("message"); %>
```

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

EXAMPLE

```
<% this.log("message"); %>
```

SESSION IMPLICIT OBJECT

In JSP, session is an implicit object of type HttpSession.This object used to set,get or remove attribute or to get session information.

EXAMPLE OF SESSION IMPLICIT OBJECT : INDEX.HTML

EXAMPLE

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

EXAMPLE

```
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
<a href="second.jsp">second jsp page</a>
```

```
%>
```

One.jsp

EXAMPLE

```
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);

%>
```

EXCEPTION IMPLICIT OBJECT

JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

EXAMPLE OF EXCEPTION IMPLICIT OBJECT

INDEX.HTML

EXAMPLE

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

EXAMPLE

```
<%
response.sendRedirect("http://www.google.com");
%>
```

APPLICATION IMPLICIT OBJECT

In JSP, application is an implicit object of type ServletContext. this application object in the Servlet programming is ServletContext.

For all jsp in a web application, there must be a single application object with application object we can share the data from one JSP to any other JSP in the web application.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

EXAMPLE OF APPLICATION IMPLICIT OBJECT

INDEX.HTML

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

WELCOME.JSP

```
<%
out.print("Welcome "+request.getParameter("uname"));
```

```
 String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

WEB.XML

```
<web-app>

<servlet>
<servlet-name>One</servlet-name>
<jsp-file>/welcome.jsp</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>One</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

PAGECONTEXT IMPLICIT OBJECT

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes.

- page
- request
- session
- application

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
<%
 String name=request.getParameter("uname");
out.print("Welcome "+name);
 pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
 <a href="One.jsp">One jsp page</a>
 %>
```

One.jsp

```
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
```

JSP DIRECTIVE ELEMENTS

The directive elements are used to do page related operation during page translation time. JSP supports 3 types of directive elements, they are;.

- Page directive
- Include directive
- Taglib directive

Syntax of directive elements

```
<@ directive attribute="value" %>
```

PAGE DIRECTIVE

The page directive defines attributes that apply to an entire JSP page. This element is used to define default of explicit operation to the jsp.

SYNTAX OF PAGE DIRECTIVE ELEMENTS

SYNTAX

```
<@ page attribute="value" %>
```

ATTRIBUTES OF JSP PAGE DIRECTIVE

- import
- contentType
- extends
- language
- buffer
- info
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding

- errorPage
- isErrorPage

JSP INCLUDE DIRECTIVE

It is used to include some other pages into the JSP document, It may be JSP file, html file or text file. This is known as static include because the target page is located and included at the time of page compilation.
The jsp page is translated only once so it will be better to include static resource.

ADVANTAGE OF INCLUDE DIRECTIVE : CODE RE-USABILITY

SYNTAX

```
<%@ include file="resourceName"%>
```

EXAMPLE

```
<html>
<body>
<%@ include file="footer.html" %>
 </body>
</html>
```

Note: The include directive includes the original content, so the actual page size grows at run-time.

Taglib Directive

This is used to use custom tags in JSP page, here the custom tag may be user defined ot JSTL tag or strust, JSF,..... etc.

Syntax

```
<@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Attributes of taglib directive

- uri

Example of JSP Taglib directive

```
<html>
<body>
 <%@ taglib uri="http://www.tutorial4us.com/customtag/tags" prefix="mytag" %>
 <mytag:currentDate/>
 </body>
</html>
```

JSP Action Element

Action elements are used to performs specific operation using JSP container, like setting values into java class, getting values from java class. The JSP action elements classified into two types are;

- JSP Standard Action Element
- JSP Custom Action Element

The standard action element followed by "JSP" prifix.

```
<jsp: standard action name>
```

Standard Action are given by JSP are;

- <jsp: include>
- <jsp: forward>
- <jsp: param>
- <jsp: params>
- <jsp: plugin>
- <jsp: useBean>

- <jsp: setProperty>
- <jsp: getProperty>