

core java

core java

by KVR

Download more materials

----- VISIT-----

<http://ameerpetmaterials.blogspot.in/>

<http://ameerpetmatbooks.blogspot.in/>

<http://satyajohnny.blogspot.in/>

Advanced JAVA (J2EE)**Day - 1:**

In IT we are developing two types of applications; they are **standalone applications** and **distributed applications**.

- A *standalone application* is one which **runs in the context of local disk**. With standalone applications we **cannot achieve the concept of data sharing**. For example C, C++, COBOL, PASCAL, etc.
- A *distributed application* is one which always **runs in the context of Browser or World Wide Web**. All distributed applications **can be accessed across the globe**. For example JAVA and DOT NET.

JAVA always provides a facility called **server independent, platform independent language**. JAVA supports a concept called **design patterns** (*design patterns* are **predefined proved rules by industry experts to avoid** side effects [recurring problems] which are occurring in software development).

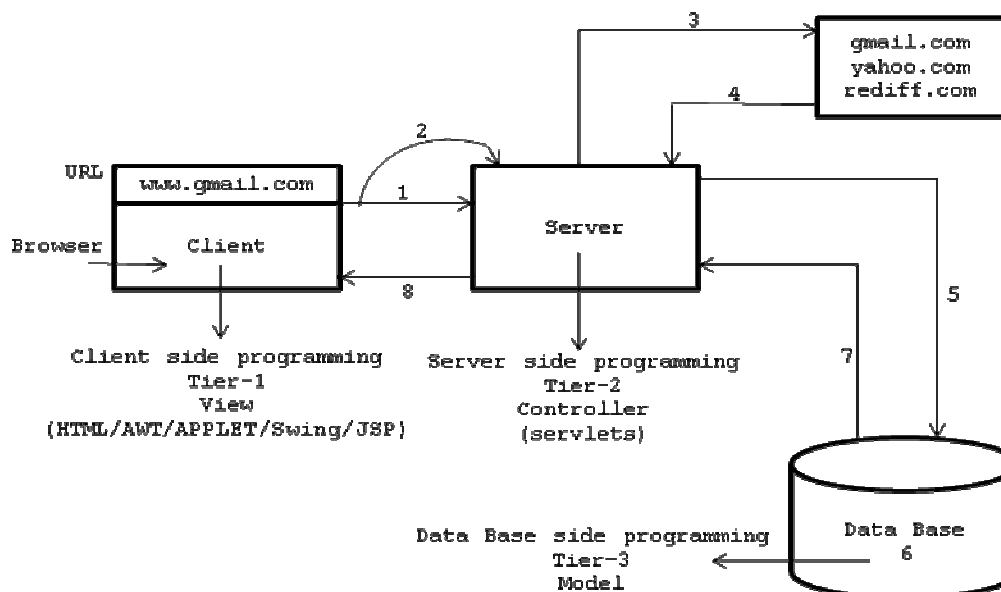
Day - 2:

In real time applications, in the case of **server side programming** one **must follow the architecture to develop a distributed application**.

To develop any *distributed application*, it is always recommended **to follow either 3-tier architecture or 2-tier architecture or n-tier architecture**.

3-tier architecture is also known as **MVC** architecture. M stands for Model (database programming), V stands for View (client side programming, HTML/AWT/APPLET/Swing/JSP) and C stands for Controller (server side programming, Servlets).

The general architecture of MVC or 3-tier:



1. Client makes a request.
2. Server side program receives the request.
3. The server looks for or search for the appropriate resource in the resource pool.

4. If the resource is not available server side program displays a user friendly message (page cannot be displayed). If the resource is available, that program will execute gives its result to server, server *interns* gives response to that client who makes a request.
5. When server want to deals with database to retrieve the data, server side program sends a request to the appropriate database.
6. Database server receives the server request and executes that request.
7. The database server sends the result back to server side program for further processing.
8. The server side program is always gives response to 'n' number of clients **concurrently**.

Day - 3:**REFLECTION**

"Reflection is the process of obtaining runtime information about the class or interface."

Runtime information is nothing but deal with the following:

1. Finding the name of the class or *interface*.
2. Finding the data members of the class or *interface*.
3. Finding number of constructors (default constructor and number of parameterized constructors).
4. Number of instance methods.
5. Number of static methods.
6. Determining modifiers of the class (**modifiers** of the class can be public, final, public + final, abstract and public + abstract).
7. Obtaining super class of a derived class.
8. Obtaining the *interfaces* which are implemented by various classes.

Real time applications of reflection:

1. Development of language compiler, debuggers, editors and browsers.
2. In order to deal with reflection in java, we must import a predefined package called *java.lang.reflect.**
3. The package *reflect* contains set of predefined classes and *interfaces* which are used by the programmer to develop reflection applications.

Number of ways to obtain runtime information about a class (or) number of ways to get an object of a class called Class:

The predefined class called *Class* is present in a package called *java.lang.Class* (**fully qualified name** of a class called *Class* is *java.lang.Class*).

In java we have 4 ways to deal with or to create an object of java.lang.Class, they are:

- 1) When we know that class name at compile time and to get runtime information about the class, we must use the following:

```
Ex1: Class c=c1.class {c1 - user defined class}
Ex2: Class c=java.lang.String.class {String - predefined class}
```

- 2) When we know the object name at runtime, to get the class name or class type of the runtime object, we must use the following:

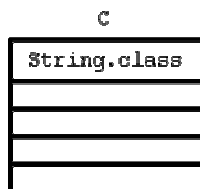
```
Ex1: cl ol=new cl ();
      Class c=ol.getClass ();
Ex2: String s=new String ("HELLO");
      Class c=s.getClass ();
```

getClass is the predefined method present in a predefined class called *java.lang.Object* and whose prototype is given below:

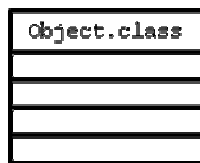
```
java.lang.Object
      |
      v
public Class getClass () → instance method
Here, public - access specifier & Class - return type
```

- 3) When an object is given at runtime, we must find runtime information about current class and its super class.

```
String s=new String ("HELLO");
Class c=s.getClass ();
```



```
Class sc=c.getSuperclass ();
      sc
```



Write a java program **to print name of the current class and its super class name?**

Answer:

```
class First
{
    public static void main (String [] args)
    {
        String s=new String ("HELLO");
        printSuperclass (s);
    }
    static void printSuperclass (Object s)
    {
        Class c=s.getClass ();
        Class sc=c.getSuperclass ();
        System.out.println ("NAME OF CURRENT CLASS : "+c.getName ());
        System.out.println ("NAME OF THE SUPER CLASS : "+sc.getName ());
    }
};
```

Output:

```
java First
NAME OF CURRENT CLASS : java.lang.String
NAME OF THE SUPER CLASS : java.lang.Object
```

Day - 4:

- 4) We know the class name at runtime and we have to obtain the runtime information about the class.

To perform the above we must use the method *java.lang.Class* and whose prototype is given below:

```
java.lang.Class  
      |  
      v  
public static Class forName (String) throws ClassNotFoundException
```

When we use the *forName* as a part of java program it performs the following operations:

- It can create an object of the class which we pass at runtime.
- It returns runtime information about the object which is created.

For example:

```
try  
{  
    Class c=Class.forName ("java.awt.Button");  
}  
catch (ClassNotFoundException cnfe)  
{  
    System.out.println ("CLASS DOES NOT EXIST...");  
}
```

forName is taking **String** as an argument. If the class is not found *forName* method throws an exception called **ClassNotFoundException**.

Here, *forName* method is a **factory method** (a *factory method* is one which return type is similar to name of the class where it presents).

Every factory method must be static and public. The class which contains a factory method is known as **Singleton** class (a java class is said to be *Singleton* class through which we can create single object per **JVM**).

For example:

java.lang.Class is called Singleton class

Write a java program **to find name of the class and its super class name** by passing the class name at runtime?

Answer:

```
class ref1  
{  
    public static void main (String [] args)  
    {  
        if (args.length==0)  
        {  
            System.out.println ("PLEASE PASS THE CLASS NAME...!");  
        }  
        else  
        {  
            try  
            {  
                Class c=Class.forName (args [0]);  
                printSuperclass (c);  
            }  
            catch (Exception e)  
            {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

```
        }
        catch (ClassNotFoundException cnfe)
        {
            System.out.println (args [0]+" DOES NOT EXISTS...");
        }
    } // else
} // main
static void printSuperclass (Class c)
{
    String s=c.getName ();
    Class sc=c.getSuperclass ();
    String sn=sc.getName ();
    System.out.println (sn+" IS THE SUPER CLASS OF "+s);
} // printSuperclass
} // ref1
```

Output:

```
java ref1 java.awt.TextField
java.awt.TextComponent IS THE SUPER CLASS OF java.awt.TextField
```

Write a java program to **print super class hierarchy** at a current class which is passed from command prompt?

Answer:

```
class Hierarchy
{
    public static void main (String [] args)
    {
        if (args.length==0)
        {
            System.out.println ("PLEASE PASS THE CLASS NAME..!");
        }
        else
        {
            try
            {
                Class c=Class.forName (args [0]);
                printHierarchy (c);
            }
            catch (ClassNotFoundException cnfe)
            {
                System.out.println (args [0]+" DOES NOT EXISTS...");
            }
        }
    }
    static void printHierarchy (Class c)
    {
        Class cl=c;
        String cname=cl.getName ();
        System.out.println (cname);
        Class sc=cl.getSuperclass ();
        while (sc!=null)
        {
            cname=sc.getName ();
            System.out.println (cname);
            cl=sc;
            sc=cl.getSuperclass ();
        }
    }
}
```

```
    }  
}  
};
```

Output:

```
java Hierarchy java.awt.TextField  
java.awt.TextField  
java.awt.TextComponent  
java.awt.Component  
java.lang.Object
```

Obtaining information about CONSTRUCTORS which are present in a class:

In order to get the constructor of the current class we must use the following method:

```
java.lang.Class  
      |  
      v  
public Constructor [] getConstructors ()
```

For example:

```
Constructor cons []=c.getConstructors ();  
System.out.println ("NUMBER OF CONSTRUCTORS = "+cons.length);
```

In order to get the parameters of the constructor we must use the following method:

```
java.lang.reflect.Constructor  
      |  
      v  
public Class [] getParameterTypes ()
```

For example:

```
Class ptype []=cons [0].getParameterTypes ();
```

Day - 5:

Write a java program to obtain constructors of a class?

Answer:

```
class ConsInfo  
{  
    public static void main (String [] args)  
    {  
        if (args.length==0)  
        {  
            System.out.println ("PLEASE PASS THE CLASS NAME..!");  
        }  
        else  
        {  
            try  
            {  
                Class c=Class.forName (args [0]);  
                printConsts (c);  
            }  
            catch (ClassNotFoundException cnfe)  
            {  
                System.out.println (args [0]+" DOES NOT EXISTS...");  
            }  
        }  
    }  
    static void printConsts (Class c)
```

```
{
    java.lang.reflect.Constructor Cons []=c.getConstructors ();
    System.out.println ("NUMBER OF CONSTRUCTORS = "+Cons.length);
    System.out.println ("NAME OF THE CONSTRUCTOR : "+c.getName());
    for (int i=0; i<Cons.length; i++)
    {
        System.out.print (c.getName ()+"(");
        Class cp []=Cons [i].getParameterTypes ();
        for (int j=0; j<cp.length; j++)
        {
            System.out.print (cp [j].getName ()+"");
        }
        System.out.println ("\b"+"");
    }
}
};
```

Obtaining METHODS information:

In order to obtain information about **methods** we must use the following methods:

```
java.lang.Class
      |
      v
Public Method [] getMethods ()
```

For example:

```
Method m []=c.getMethods ();
System.out.println ("NUMBER OF METHODS = "+m.length);
```

Associated with methods we have return type of the method, name of the method and types of parameters passed to a method.

The Method class contains the following methods:

1. public Class *getReturnType* ();
2. public String *getName* ();
3. public Class [] *getParameterTypes* ();

Method-1 gives return type of the method, Method-2 gives name of the method and Method-3 gives what parameters the method is taking.

Write a java program **to obtain information about methods** which are present in a class?

Answer:

```
import java.lang.reflect.*;
class MetInfo
{
    public static void main (String [] args)
    {
        try
        {
            if (args.length==0)
            {
                System.out.println ("PLEASE PASS THE CLASS NAME..!");
            }
            else
            {
                Class c=Class.forName (args [0]);
                printMethods (c);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```



```
        }
    }
    catch (ClassNotFoundException cnfe)
    {
        System.out.println (args [0]+" DOES NOT EXISTS...");
    }
}
static void printMethods (Class c)
{
    Method m []=c.getMethods ();
    System.out.println ("NUMBER OF METHODS = "+m.length);
    System.out.println ("NAME OF THE CLASS : "+c.getName ());
    for (int i=0; i<m.length; i++)
    {
        Class cl=m [i].getReturnType ();
        String rtype=cl.getName ();
        String mname=m [i].getName ();
        System.out.print (rtype+" "+mname+"(");
        Class mp []=m [i].getParameterTypes ();
        for (int j=0; j<mp.length; j++)
        {
            String ptype=mp [j].getName ();
            System.out.print (ptype+",");
        }
        System.out.println ("\b"+"")");
    }
}
};
```

Obtaining FIELDS or DATA MEMBERS of a class:

In order to obtain information about **fields** or **data members** of the class we must use the following method.

```
java.lang.Class
      |
      v
Field [] getFields ()
```

For example:

```
Field f []=c.getFields ();
System.out.println ("NUMBER OF FIELDS = "+f.length);
```

Associated with field or data member there is a data type and field name:

The Field class contains the following methods:

1. public Class *getType* ();
2. public String *getName* ();

Method-1 is used for obtaining data type of the field and Method-2 is used for obtaining name of the field.

Write a java program **to print fields or data members** of a class?

Answer:

```
import java.lang.reflect.Field;
class Fields
{
    void printFields (Class c)
    {
```

```
Field f []=c.getFields ();
System.out.println ("NUMBER OF FIELDS : "+f.length);
for (int i=0; i<f.length; i++)
{
    String fname=f [i].getName ();
    Class s=f [i].getType ();
    String ftype=s.getName ();
    System.out.println (ftype+" "+fname);
}
};
class FieldsDemo
{
    public static void main (String [] args)
    {
        if (args.length==0)
        {
            System.out.println ("PLEASE PASS THE CLASS NAME..!");
        }
        else
        {
            try
            {
                Class c=Class.forName (args [0]);
                Fields fs=new Fields ();
                fs.printFields (c);
            }
            catch (ClassNotFoundException cnfe)
            {
                System.out.println (args [0]+"NOT FOUND...");
            }
        }
    }
};
```

Day - 6:

JDBC (Java Database Connectivity)

“JDBC is a kind of specification developed by SUN Microsystems to store the data permanently”.

In Information Technology we have two approaches’ to store the data permanently. They are **through files** and **through database**.

Whatever data we store permanently in the form of a file, the file will not provide enough security to the data from unauthorized users.

In order to save or store the data permanently in the form of a file we must use the concept of serialization.

Serialization: *Serialization* is the mechanism of saving the state of the object permanently in the form of a file.

Steps for developing *SERIALIZABLE* SUB CLASS:

A *Serializable* sub class is one which implements a predefined interface called *java.io.Serializable*

1. Choose the appropriate package to keep *Serializable* sub class.
2. Choose the user defined class whose object participates in *Serializable* process.
3. Every user defined class must implements a predefined *interface* called *Serializable*.
4. Choose the set of data members for *Serializable* sub class.
5. Develop the set of set methods for each and every data members of the class.
6. Develop the set of get methods for each and every data members of the class.

The above class is known as **java bean class** or **component style based programming** or **POJO (Plain Old Java Object) class**.

For example:

```
package ep; //step-1
import java.io.*;
public class Emp implements Serializable // Emp (step-2) & Serializable (step-3)
{
    int empno;
    String ename;
    float sal;
    // above data members (step-4)
    public void setEmpno (int empno)
    {
        this.empno=empno;
    }
    public void setEname (String ename)
    {
        this.ename=ename;
    }
    public void setSal (float sal)
    {
        this.sal=sal;
    } // above set methods (step-5)
    public int getEmpno ()
    {
        return empno;
    }
    public String getEname ()
    {
        return ename;
    }
    public float getSal ()
    {
        return sal;
    } // above get methods (step-6)
};
```

Day - 7:

Serializable process:

Steps for SERIALIZATION process:

1. Create an object of *Serializable* sub class.

For example:

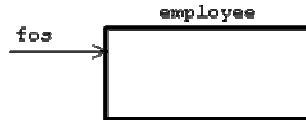
```
sp.emp eo=new sp.emp ();
```

2. Call set of set methods to place user defined values in a *Serializable* sub class object.

For example:

```
eo.setEmpno (10);  
eo.setName ("KVR");  
eo.setSal (10000.00f);
```

3. Choose the file name and open it *into* write mode or output mode with the help of *java.io.FileOutputStream*

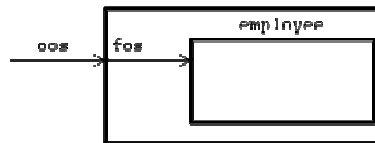


For example:

```
FileOutputStream fos=new FileOutputStream ("employee");
```

4. Since an object of *FileOutputStream* cannot write the entire object at a time to the file. Hence, it is recommended to use a predefined class called *ObjectOutputStream* class. *ObjectOutputStream* class contains the following constructor which takes an object of *FileOutputStream* class.

```
ObjectOutputStream  
└──  
ObjectOutputStream (FileOutputStream);
```



For example:

```
ObjectOutputStream oos=new ObjectOutputStream (fos);
```

5. In order to write the entire object at a time to the file *ObjectOutputStream* contains the following method:

```
ObjectOutputStream  
└──  
public void writeObject (Object);
```

For example:

```
oos.writeObject (eo);
```

6. Close the files which are opened in write mode.

For example:

```
oos.close ();  
fos.close ();
```

Write a java program which will save the *Serializable* sub class object *into* a file?

Answer:

```
import ep.Emp;  
import java.io.*;  
class serp  
{  
    public static void main (String [] args) throws Exception
```

```
{
    Emp eo=new Emp ();
    eo.setEmpno (100);
    eo.setName ("KVR");
    eo.setSal (10000.00f);
    FileOutputStream fos=new FileOutputStream ("employee");
    ObjectOutputStream oos=new ObjectOutputStream (fos);
    oos.writeObject (eo);
    System.out.println ("EMPLOYEE OBJECT SAVED SUCCESSFULLY...");
    oos.close ();
    fos.close ();
}
};
```

Output:

EMPLOYEE OBJECT SAVED SUCCESSFULLY...

De-Serializable process: It is the process of retrieving the record from the file *into* main memory of the computer.

Steps for DE-SERIALIZATION process:

1. Create an object of *Serializable* sub class.

For example:

```
emp eol=new emp ();
```

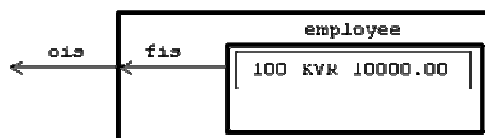
2. Choose the file name and open it *into* read mode with the help of *FileInputStream* class.

**For example:**

```
FileInputStream fis=new FileInputStream ("employee");
```

3. Since an object of *FileInputStream* cannot read the entire object at a time from the file. Hence, it is recommended to create an object of *ObjectInputStream* class. *ObjectInputStream* class contains the following constructor which takes object of *FileInputStream* as a parameter.

```
ObjectInputStream  
↓  
ObjectInputStream (FileInputStream);
```

**For example:**

```
ObjectInputStream ois=new ObjectInputStream (fis);
```

4. *ObjectInputStream* class contains the following method which will read the entire object at a time where ever *ObjectInputStream* is pointing.

```
ObjectInputStream  
    ↓  
public Object readObject ();
```

For example:

```
Object obj=ois.readObject ();
```

5. An object of Object does not contain set of get methods and they are defined in its sub class called emp. Hence, an object of Object must be type casted to *Serializable* sub class object.

For example:

```
eo1= (emp) obj;
```

6. Apply set of get methods to obtain the data from de-serialized object i.e., eo1.

For example:

```
System.out.println ("empno : "+eo1.getEmpno ());  
System.out.println ("empname : "+eo1.getEname ());  
System.out.println ("empsal : "+eo1.getSal ());
```

7. Close the files which are opened in read mode or input mode.

For example:

```
ois.close ();  
fis.close ();
```

Write a java program which will de-*Serializable* from the specified file?

Answer:

```
import ep.Emp;  
import java.io.*;  
class dserp  
{  
    public static void main (String [] args) throws Exception  
    {  
        Emp eo1=new Emp ();  
        FileInputStream fis=new FileInputStream ("employee");  
        ObjectInputStream ois=new ObjectInputStream (fis);  
        Object obj=ois.readObject ();  
        eo1= (Emp) obj;  
        System.out.println ("EMP NO : "+eo1.getEmpno ());  
        System.out.println ("EMP NAME : "+eo1.getEname ());  
        System.out.println ("EMP SALARY : "+eo1.getSal ());  
        ois.close ();  
        fis.close ();  
    }  
};
```

Output:

```
EMP NO : 100  
EMP NAME : KVR  
EMP SAL : 10000.0
```

Day - 8:

When we don't want the variables to participate in serialization process, which type of variables must be made it as transient i.e., transient variables will not participate in serialization process.

In general we have four types of serializations, they are:

1. **Complete serialization:**

It is one in which all the data members of the class will participate in serialization process.

2. **Selective serialization:**

It is one in which selective data members of the class (non-transient variables) will participate in serialization process.

3. **Manual serialization:**

It is one in which the derived class explicitly implements a predefined *interface* called *java.io.Serializable*. The *interface Serializable* does not contain any abstract methods and this type of *interface* is known as marked or tagged *interface*.

4. **Automatic serialization:**

It is one in which the user defined derived class extends sub class of *Serializable interface*.

For example:

```
class Bank extends Emp
{
    ..... ;
    ..... ;
};
```

In real world application we cannot store the data permanently in the form of files. Since, a file does not provide any security to prevent unauthorized modifications. Hence, it is recommended to store the data permanently in the form of database.

JDBC:

In the *initial* days of database technology various database vendors has developed various database products. Anybody who want to deal with any database the programmer must have complete knowledge about the database which they are using i.e., in the *initial* days all the databases are available with a specific library (native library) which was developed in 'C' language. In the context the programmer must have complete knowledge about the native library of the database which is a complex process.

In later stages all database vendors gathered and developed XOPEN/CLI (Call Level *Interface*) software along with Microsoft which is known as ODBC (Open Database Connectivity).

ODBC is having a common API or library for various databases and it is also developed in 'C' language and it is a platform dependent.

In later stages SUN micro systems has developed a general specification called *JDBC* which contains a common API for all databases with platform independent.

In order to deal with any database to represent the data permanently, we must use **driver** (a *driver* is a software which acts as a middle layer between database and front end application i.e., java) of the specific database. In real world we have various drivers are available for various database products.

Types of DRIVERS:

SUN micro systems has divided various database drivers of various database products *into* four types, they are:

1. Type-1 (*JDBC-ODBC bridge driver*).

2. Type-2 (Native or partial java drivers).
3. Type-3 (Net protocol or *intermediate* database server access drivers) and
4. Type-4 (Thin drivers or pure drivers or all java drivers).

Steps for developing a JDBC program:

1. Loading the drivers.
2. Obtain the connection or specify the URL.
3. Pass the query.
4. Process the result which is obtained from database.
5. Close the connection.

Day - 9:

JDBC is the standard specification released by SUN micro systems to develop the applications in database world. *JDBC* contains set of *interfaces* and these *interfaces* are implemented by various database vendors and server vendors.

A **driver** is nothing but a java class which acts as a middle layer between java program and database program. As on today all the drivers are developed by either database vendors or server vendors.

For example:

```
class x implements ____  
{  
    .....;  
    .....;  
    .....;  
};
```

Here, **x** is driver and ____ is *JDBC interface*.

In database world, each and every database vendor has developed their drivers and released to the market in the form of jar files.

TYPE-1 DRIVERS

These are developed by SUN micro systems. The name of the Type-1 driver is ***JdbcOdbcDriver***. The driver *JdbcOdbcDriver* is found in a package called *sun.jdbc.odbc*. Using this driver we can develop **only 2-tier** applications (a java program and database). This type of driver is purely implemented in 'C' language and this driver is platform dependent in nature.

Loading the drivers:

Loading the drivers is nothing but creating an object of appropriate Driver class. In order to load the drivers we have two ways, they are:

1) Using *Class.forName***For example:**

```
Class.forName (Sun.jdbc.odbc.JdbcOdbcDriver);  
Class.forName (oracle.jdbc.driver.OracleDriver);
```

2) Using *DriverManager.registerDriver*

DriverManager class contains the following method which will load the driver at runtime.

For example:

```
public static void registerDriver (java.sql.Driver);
```

Driver is an interface which is implemented by various database vendors and server vendors. If the appropriate driver object is created that driver object will act as a middle layer between program and database. If the driver is not found we get an exception called **java.sql.SQLException**

For example:

```
DriverManager.registerDriver (new Sun.jdbc.odbc.JdbcOdbcDriver);  
[for oracle driver is – classes111.jar]
```

Day - 10:

How to obtain the connection: After loading the drivers, it is required to obtain the connection from the database.

Syntax or URL for obtaining connection:

```
      Main Protocol : Sub Protocol : DataSourceName  
      jdbc          : odbc         : oracle
```

Here, *jdbc* is the main protocol which takes java request and hand over into database environment through Data Source Name. *odbc* is the sub protocol which takes the database result and gives to java environment. **Data Source Name** is the configuration tool in the current working machine through which the data is passing from java environment to database and database to java environment.

In order to obtain the connection from the database, as a part of *jdbc* we have a predefined class called *java.sql.DriverManager* which contains the following methods:

1. `public static Connection getConnection (String URL);`
2. `public static Connection getConnection (String URL, String username, String password);`

Method-1, we use to obtain the connection from those databases where there is no username and password databases. Method-2 is used for those databases where there is username and password.

For example:

```
Connection con1=DriverManager.getConnection ("jdbc : odbc : Access");  
Connection con2=DriverManager.getConnection ("jdbc : odbc : oracle","scott","tiger");
```

Pass the query: A query is nothing but a request or question to the database.

Queries are of three types; they are **static query, dynamic** or **pre-compiled query** and **stored procedures**.

STATIC QUERY: *Static query* is one in which the data is passed in the query itself.

For example:

1. `select * from Student where marks>50;`
2. `insert into Student values (100, 'abc', 90.86);`

In order to execute static queries we must obtain an object of *java.sql.Statement* interface. In the *interface java.sql.Connection* we have the following method for obtaining an object of *Statement* interface.

```
java.sql.Connection ─┐  
                      │  
                      ▼  
public Statement createStatement ();
```

For example:

```
Statement st=con1.createStatement ();
```

On database three categories of operations takes place, they are **insertion**, **deletion** and **updation**. In order to perform these operations we must use the following method which is present in Statement *interface*.

```
java.sql.Statement ─┐  
                    │  
                    ▼  
public int executeUpdate (String);
```

Here, *String* represents either static insertion or static deletion or static updation. The return type *int* represents the status of the query. If the query is not successful it returns **zero** and if the query is successful it returns **non-zero**.

Write a *jdbc* program which will **insert a record in** the Student database?

Answer:

```
import java.sql.*;  
class InsertRec  
{  
    public static void main (String [] args)  
    {  
        try  
        {  
            Driver d=new Sun.jdbc.odbc.JdbcOdbcDriver ();  
            DriverManager.registerDriver (d);  
            System.out.println ("DRIVERS LOADED...");  
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");  
            System.out.println ("CONNECTION ESTABLISHED...");  
            Statement st=con.createStatement ();  
            int i=st.executeUpdate ("insert into student values (10,'suman',60.87);");  
            System.out.println (i+" ROWS SELECTED...");  
            con.close ();  
        }  
        catch (Exception e)  
        {  
            System.out.println ("DRIVER CLASS NOT FOUND...");  
        }  
    }  
};
```

Day - 11:**Processing the query result:**

In order to **execute the select statement** or in order to **retrieve the data** from database we must use the following method which is present in *java.sql.Statement* interface.

```

java.sql.Statement
      |
      v
public ResultSet executeQuery (String);

```

Here, *String* represents a query which contains select statement. *executeQuery* returns an object of *ResultSet* to hold the number of records returned by select statement. *ResultSet* is an interface whose object contains all the records returned by a query and it will point to just before the first record.

ResultSet object
rs →

stno	stname	marks
1	a	10
2	b	20
3	c	30

For example:

```
ResultSet rs=st.executeQuery ("select * from Student");
```

The *ResultSet* object is pointing by default just before the first record, in order to bring first record we must use the below given method. Method returns true when rs contains next record otherwise it returns false.

```
public boolean next ();
```

In order to obtain the data of the record (collection of field values) we must use the following method:

```
public String getString (int colno);
```

Whatever the data retrieving from the record that data will be treated as string data.

For example:

```
String s1=rs.getString (1);
String s1=rs.getString (2);
String s1=rs.getString (3);
```

Write a java program to retrieve the data from emp database?

Answer:

```

import java.sql.*;
class SelectData
{
    public static void main (String [] args) throws Exception
    {
        DriverManager.registerDriver (new Sun.jdbc.odbc.JdbcOdbcDriver ());
        System.out.println ("DRIVERS LOADED...");
        Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");
        System.out.println ("CONNECTION ESTABLISHED...");
        Statement st=con.createStatement ();
        ResultSet rs=st.executeQuery ("select * from dept");
        while (rs.next ())
        {
            System.out.println (rs.getString (1)+" "+rs.getString (2)+" "+rs.getString (3));
        }
        con.close ();
    }
}

```

```

    }
};

```

ResultSet:

1. An object of *ResultSet* allows us to retrieve the data by default only in **forward direction** but not in backward direction and random retrieval.
2. Whenever we get the database data in *ResultSet* object which will be temporarily disconnected from the database.
3. *ResultSet* object **does not** allows us to perform any modifications on *ResultSet* object.
4. Hence, the *ResultSet* is **by default non-scrollable** disconnected *ResultSet*.

DYNAMIC or PRE-COMPILED QUERIES:

1. Dynamic queries are those for which the data is passed at runtime.
2. To execute dynamic queries we must obtain an object of *PreparedStatement*.

Differentiate between Statement and *PreparedStatement*?

Answer:

Statement	<i>PreparedStatement</i>
1. This <i>interface</i> is used for executing static queries.	1. This <i>interface</i> is used for executing dynamic queries.
2. When we execute static queries with respect to Statement object; compilation, parsing and execution of the query takes place each and every time.	2. When we execute dynamic queries using <i>PreparedStatement</i> object; compilation, parsing and execution of the query takes place first time and from second time onwards only execution phase takes place.
3. There is a possibility of losing performance of a <i>jdbc</i> program. Since, compilation, parsing and execution taking place each and every time.	3. We can get the performance of <i>jdbc</i> program. Since, compilation and parsing takes place only one time.

In order to obtain an object of *PreparedStatement* we must use the following method:

```

java.sql.Connection
    |
    v
public PreparedStatement prepareStatement (String);

```

Here, String represents dynamic query.

For example:

```
PreparedStatement ps=con.prepareStatement ("select * from dept where deptno=?");
```

Here, the '?' is known as **dynamic substitution operator** or **positional parameter**. The position of the positional parameters must always starts from left to right with the numbers 1, 2.....n.

In order to **set the values to the positional parameters**, we must use the following methods which are present in prepared statement *interface*.

```
public void setByte (int, byte);
public void setShort (int, short);
public void setInt (int, int);
public void setLong (int, long);
public void setFloat (int, float);
public void setDouble (int, double);
public void setChar (int, char);
public void setString (int, string);
```

In general *PreparedStatement* interface contains the following generalized method to set the values for positional parameters.

```
public void setXXX (int, XXX);
```

Here, *int* represents position number of the positional parameter. XXX represents value of either fundamental data type or **string or date**.

For example:

```
ps.setInt (1, 10);
```

In order to execute the DCL statements (select) and DML statements (insert, delete and update) we must use the following methods which are present in *PreparedStatement* interface.

```
public int executeUpdate (); → Dynamic DML/DDL
public ResultSet executeQuery (); → Dynamic DCL (select)
```

For example:

```
ResultSet rs=ps.executeQuery ();
```

Close connection

For example:

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

Day - 12:

Write a java program to insert a record in dept database by accepting the data from keyboard at **runtime using dynamic queries?**

Answer:

```
import java.sql.*;
import java.io.*;
class InsertRecRun
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
            System.out.println ("DRIVERS LOADED...");
```

```
        Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
BudDinu","scott","tiger");
        System.out.println ("CONNECTION OBTAINED...");
        PreparedStatement ps=con.prepareStatement ("insert into dept values (?,?,?)");
        DataInputStream dis=new DataInputStream (System.in);
        System.out.println ("ENTER DEPARTMENT NUMBER : ");
        String s1=dis.readLine ();
        int dno=Integer.parseInt (s1);
        System.out.println ("ENTER DEPARTMENT NAME : ");
        String dname=dis.readLine ();
        System.out.println ("ENTER LOCATION NAME : ");
        String loc=dis.readLine ();
        ps.setInt (1, dno);
        ps.setString (2, dname);
        ps.setString (3, loc);
        int i=ps.executeUpdate ();
        System.out.println (i+"ROW(s) INSERTED...");
        con.close ();
    }
    catch (Exception e)
    {
        e.printStackTrace ();
    }
}
// main
}; // InsertRecRun
```

Write a java program to retrieve the records from a specified database by accepting input from keyboard?

Answer:

```
import java.sql.*;
import java.io.*;
class SelectDataRun
{
    public static void main (String [] args)
    {
        try
        {
            Class.forName ("Sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");
            System.out.println ("CONNECTION ESTABLISHED...");
            PreparedStatement ps=con.prepareStatement ("select * from dept where deptno");
            DataInputStream dis=new DataInputStream (System.in);
            System.out.println ("ENTER DEPARTMENT NUMBER : ");
            String s1=dis.readLine ();
            int dno=Integer.parseInt (s1);
            ps.setInt (1, dno);
            ResultSet rs=ps.executeQuery ();
            while (rs.next ())
            {
                System.out.print (rs.getString (1)+" "+rs.getString (2)+" "+rs.getString (3));
            }
            con.close ();
        }
        catch (Exception e)
```

```
        {  
            e.printStackTrace ();  
        }  
    }  
    }  
}; // SelectDataRun
```

TYPE – 4 DRIVERS

In order to avoid the disadvantages of Type-1 drivers, we have to deal with Type-4 drivers.

Disadvantages of Type-1:

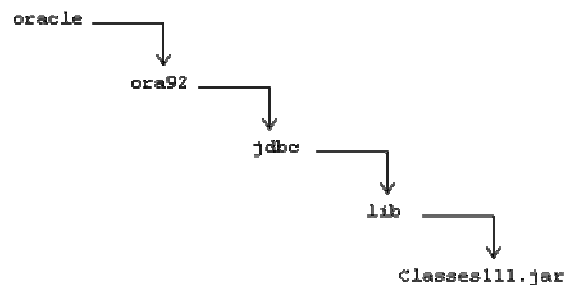
1. Since, it is developed in 'C' language; this type of driver is treated as platform dependent.
2. These are a waste of memory space. Since, we are creating a DSN (**Data Source Name**) for each and every database connection and it leads to less performance.
3. We are unable to develop 3-tier applications.

Advantages of Type-4:

1. This driver gives effective performance for every *jdbc* application. Since, there is no DSN.
2. Since, this driver is developed in java by database vendors, internally JVM need not to convert platform dependent to platform independent.

The only **disadvantage of Type-4** is we are unable to develop 3-tier applications.

Type-4 drivers are supplied by Oracle Corporation by developing *into* java language. OracleDriver is the name of Type-4 driver which is released by Oracle Corporation in the form of *classes111.jar*



When we want to make use of Type-4 driver as a part of a java program, we must first set classpath for oracle driver by using the following:

```
set CLASSPATH=C:\oracle\ora92\jdbc\lib\classes111.jar; .;
```

For example:

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

In order to obtain the connection from oracle database we must follow the following syntax:

Main protocol	: Sub protocol	: Name of the driver	:@ IP address or DSN	: Port no.	: Service ID
↓	↓	↓	↓	↓	↓
<code>jdbc</code>	<code>oracle</code>	<code>thin</code>	<code>:@</code>	<code>Localhost/ 127.0.0.1</code>	<code>1521</code> <code>AshaKrishna</code>

For example:

```
Connection con=DriverManager.getConnection ("jdbc : oracle : thin :@ localhost : 1521 : AshaKrishna", "scott", "tiger")
```

In order to obtain port number and service ID of oracle database we must look for **tnsnames.ora** which is found in **C:\oracle\ora92\network\admin**

Day - 13:**STORED PROCEDURES:**

In general, we are performing the database operations by using ordinary SQL statements. When we want to execute n number of SQL statements through java program, the java environment is executing **those queries one at a time which leads to lack of performance** to a *jdbc* application.

In order to improve the performance of *jdbc* application, it is recommended to write all n number of SQL statements in a single program (in case of oracle it is called PL/SQL program) and that **program will execute at a time irrespective of number** of SQL statements which improves the performance of a java application.

A program which contains n number of SQL statements and residing a database environment is known as **stored procedure**.

Stored procedures are divided *into* two types, they are **procedure** and **function**.

- A *procedure* is one which contains block of statements which will return either **zero** or more than **one** value.

Syntax for creating a procedure:

```
create or replace procedure <procedure name> (parameters if any)
as/is
    local variables;
begin
    block of statements;
end;
/
```

In order to call a procedure from java environment we must call on the name of procedure.

For example:

```
create or replace procedure procl
as
    i out number;
    a out number;
    b number;
    c number;
    x in out number;
begin
    i:=40+42;
    b:=10;
    c:=20;
    a:=b+c;
    x:=x+b+c;
end;
/
```


Create an oracle procedure which takes two input numbers and it must return sum of two numbers, multiplication and subtraction?

Answer:

```
create or replace procedure proc2 (a in number, b number, n out number, n2 out
number, n3 out number)
as
begin
    n1:=a+b;
    n2:=a*b;
    n3:=a-b;
end;
/
```

- A *function* is one which contains n number of block of statements to perform some operation and it returns a **single value** only.

Syntax for creating a function:

```
create or replace function (a in number, b in number) return <return type>
as
    n1 out number;
begin
    n1:=a+b;
    return (n1);
end;
/
```

In order to execute the stored procedures from *jdbc* we must follow the following steps:

1. Create an object of **CallableStatement** by using the following method:

```
java.sql.Connection -----
                             |
                             v
public CallableStatement prepareCall (String);
```

Here, **String** represents a call for calling a stored procedure from database environment.

2. Prepare a call either for **a function** or for **a procedure** which is residing in database.

Syntax for calling a function:

"{? = call <name of the function> (?, ?, ?...)}"

For example:

```
CallableStatement cs=con.prepareCall ("{? = call fun1 (?, ?)}");
```

The **positional** parameters numbering will always from left to right starting from 1. In the above example the positional parameter-1 represents **out** parameter and the positional parameter-2 and parameter-3 represents **in** parameters.

Syntax for calling a procedure:

"{call <name of the procedure> (?, ?, ?...)}"

For example:

```
CallableStatement cs=con.prepareCall ("{call fun1 (?, ?, ?, ?, ?)}");
```

3. Specify which input parameters are by using the following generalized method:

```
Public void setXXX (int, XXX);
```

For example:

```
cs.setInt (2, 10);
```

```
cs.setInt (3, 20);
```

4. Specify which output parameters are by using the following generalized method:

```
java.sql.CallableStatement  

    ↓  

public void registerOutParameter (int, jdbc data type for equivalent database datatype);
```

In *jdbc* we have a predefined class called `java.sql.Types` which contains various data types of *jdbc* which are equivalent to database data types.

Java	Jdbc	Database
<i>int</i>	<i>INTEGER</i>	number
String	VARCHAR	varchar2
Short	TINY <i>INTEGER</i>	number
Byte	SMALL <i>INTEGER</i>	number

All the data members which are available in `Types class` are belongs to **public static final data members**.

For example:

```
cs.registerOutParameter (1, Types.INTEGER);
```

5. Execute the stored procedure by using the following method:

```
java.sql.CallableStatement  

    ↓  

public boolean execute ();
```

For example:

```
cs.execute ();
```

6. Get the values of out parameters by using the following method:

```
public XXX getXXX (int);
```

Here, *int* represents position of out parameter. XXX represents fundamental data type or string or date.

For example:

```
int x=cs.getInt (1);
```

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

Day - 14:

Write a java program which illustrates the concept of function?

Answer:

StuFun:

```
create or replace function StuFun  

(a in number, b in number, n1 out number) return number  

as
```

```
        n2 number;
begin
    n1:=a*b;
    n2:=a+b;
    return (n2);
end;
/
```

FunConcept.java:

```
import java.sql.*;
import java.io.*;
class FunConcept
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
            BudDinu","scott","tiger");
            System.out.println ("CONNECTION OBTAINED...");
            DataInputStream dis=new DataInputStream (System.in);
            System.out.println ("ENTER FIRST NUMBER : ");
            String s1=dis.readLine ();
            System.out.println ("ENTER SECOND NUMBER : ");
            String s2=dis.readLine ();
            int n1=Integer.parseInt (s1);
            int n2=Integer.parseInt (s2);
            CallableStatement cs=con.prepareCall (" {?=call ArthFun (?,?,?) }");
            cs.setInt (2, n1);
            cs.setInt (3, n2);
            cs.registerOutParameter (1, Types.INTEGER);
            cs.registerOutParameter (4, Types.INTEGER);
            cs.execute ();
            int res=cs.getInt (1);
            int res1=cs.getInt (4);
            System.out.println ("SUM OF THE NUMBERS : "+res);
            System.out.println ("MULTIPLICATION OF THE NUMBERS : "+res1);
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
    } // main
} // FunConcept
```

Write a java program which illustrates the concept of procedure?

Answer:**StuPro:**

```
create or replace procedure StuPro
(no in number, name in varchar2, loc1 out varchar2)
as
begin
    select dname, loc into name, loc1 from dept
```

```
        where deptno=no;
        insert int abc values (no, name, loc1);
end;
/
```

ProConcept.java:

```
import java.sql.*;
import java.io.*;
class ProConcept
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
BudDinu","scott","tiger");

            DataInputStream dis=new DataInputStream (System.in);
            System.out.println ("ENTER DEPARTMENT NUMBER : ");
            String s1=dis.readLine ();
            int n1=Integer.parseInt (s1);
            CallableStatement cs=con.prepareCall ("{call StuPro (?,?,?)");
            cs.setInt (1,n1);
            cs.registerOutParameter (2, Types.VARCHAR);
            cs.registerOutParameter (3, Types.VARCHAR);
            cs.execute ();
            String res=cs.getString (2);
            String res1=cs.getString (3);
            System.out.println ("DEPARTMENT NAME : "+res);
            System.out.println ("DEPARTMENT LOCATION : "+res1);
        }
        catch (Exception e)
        {
            System.out.println (e);
        }
    }
} // main
} // ProConcept
```

Retrieving the data from CONVENTIONAL DATABASE (MS-Excel):

In real world applications, there is a possibility of retrieving the data from conventional data bases like ms-excel.

Steps for retrieving data from ms-excel:

1. Create an excel sheet, enter the column names along with data, **rename the sheet1** as user defined name which is treated as table name.

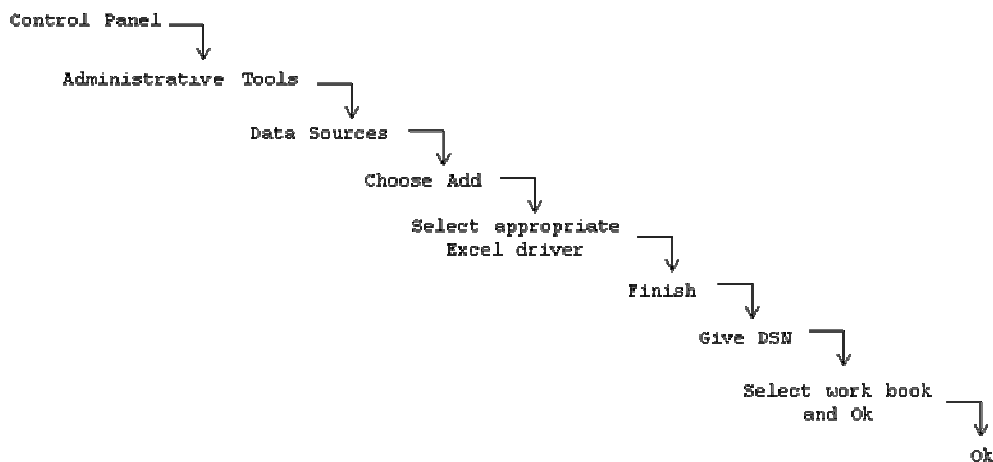
stno	stname	marks
10	suman	90.99
20	kalyan	60.87
30	oduri	77.77
Sheet1\Sheet2\Sheet3\		

2. Save the excel sheet in a current working directory.

For example:

D:\advanced\jdbc\stbook.xls

3. Create DSN for excel



4. Use xldsn while obtaining a connection from excel.

For example:

```
Connection con=DriverManager.getConnection ("jdbc : odbc : xldsn");
```

NOTE: In order to refer excel sheet name as a database sheet name we should use the format [sheet name> \$]

Write a *jdbc* program to retrieve the data from excel?

Answer:

```

import java.sql.*;
class XSelect
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new Sun.jdbc.odbc.JdbcOdbcDriver ());
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:odbc:xldsn");
            System.out.println ("CONNECTION ESTABLISHED...");
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from [student$]");
        }
    }
}

```

```
        while (rs.next ())
        {
            System.out.println (rs.getString (1)+" "+rs.getString (2)+" "+rs.getString (3));
        }
        con.close ();
    }
    catch (SQLException sqle)
    {
        sqle.printStackTrace ();
    }
}
} // main
}; // XSelect
```

Day - 15:

Metadata:

Data about data is known as metadata. Metadata can be obtained at two levels, they are **user database details** and **universal database details**.

Obtaining user database details:

In order to obtain user database details we must follow the following procedure:

1. Obtain an object of *ResultSetMetaData* by using the following method which is present in **ResultSet**.

```
java.sql.ResultSet
      |
      v
public ResultSetMetaData getMetaData ();
```

For example:

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

2. In general every user database contains number of columns, name of the columns and type of columns. In order to obtain the above information we must use the following methods which are present in *ResultSetMetaData* interface.

```
public int getColumnCount ();
public String getColumnName ();
public String getColumnLabel ();
public String getColumnType ();
```

Obtaining universal database details:

When we get a connection from the database we can come to know which database we are using. To obtain information about universal database we must use the following steps:

1. Obtain an object of *DatabaseMetaData* by calling the following method which is present in **Connection interface**.

```
java.sql.Connection
      |
      v
public DatabaseMetaData getMetaData ();
```

For example:

```
DatabaseMetaData dmd=con.getMetaData ();
```

2. In general every universal database contains database name, database version, driver name, driver version, driver major version and driver minor version. To obtain these information, **DatabaseMetaData** interface contains the following methods:

```
public String getDatabaseProductName ();
public String getDatabaseProductVersion ();
public String getDriverName ();
public String getDriverVersion ();
public String getDriverMajorVersion ();
public String getDriverMinorVersion ();
```

Write a java program which illustrates the concept of *DatabaseMetaData* and *ResultSetMetaData*?

Answer:

```
import java.sql.*;
class MetaData
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new Sun.jdbc.odbc.JdbcOdbcDriver ());
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc : odbc : oradsn","scott","tiger");
            System.out.println ("CONNECTION ESTABLISHED...");

            // UNIVERSAL DATABASE DETAILS
            DatabaseMetaData dmd=con.getMetaData ();
            System.out.println ("DATABASE NAME : "+dmd.getDatabaseProductName ());
            System.out.println ("DATABASE VERSION : "+dmd.getDatabaseProductVersion ());
            System.out.println ("NAME OF THE DRIVER : "+dmd.getDriverName ());
            System.out.println ("VERSION OF THE DRIVER : "+dmd.getDriverVersion ());
            System.out.println ("MAJOR VERSION OF DRIVER : "+dmd.getDriverMajorVersion ());
            System.out.println ("MINOR VERSION OF DRIVER : "+dmd.getDriverMinorVersion ());

            // USER DATABASE DETAILS
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from dept");
            ResultSetMetaData rsmd=rs.getMetaData ();
            System.out.println ("NUMBER OF COLUMNS : "+rsmd.getColumnCount ());
            for (int i=1; i<=rsmd.getColumnCount (); i++)
            {
                System.out.println ("NAME OF THE COLUMN : "+rsmd.getColumnName (i));
                System.out.println ("TYPE OF THE COLUMN : "+rsmd.getColumnType (i));
            }
            con.close ();
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
}
// main
}; // MetaData
```

Write a java program which *points* the data of a table along with its column names?

Answer:

```
import java.sql.*;
class Table
{
    public static void main (String [] args)
    {
        try
        {
            DriverManager.registerDriver (new Sun.jdbc.odbc.JdbcOdbcDriver ());
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");
            System.out.println ("CONNECTION ESTABLISHED...");
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from dept");
            ResultSetMetaData rsmd=rs.getMetaData ();
            System.out.println ("=====");

            // PRINTING COLUMN NAME
            for (int i=1; i<=rsmd.getColumnCount (); i++)
            {
                System.out.print (rsmd.getColumnName (i)+" ");
            }
            System.out.println ("");
            System.out.println ("=====");

            // PRINTING THE DATA OF THE TABLE
            while (rs.next ())
            {
                for (int j=1; j<=rsmd.getColumnCount (); j++)
                {
                    System.out.print (rs.getString (j)+" ");
                }
                System.out.println ("");
            }
            con.close ();
        }
        catch (SQLException sqle)
        {
            sqle.printStackTrace ();
        }
    }
}
// main
}; // Table
```

Day - 16:

Developing flexible *jdbc* applications:

When we write any *jdbc* application, we use to specify the specific details regarding driver names, URL, which database we are using and table names. When we want to change the *jdbc* application for some other details regarding driver name, URL, etc., we must change *into* java program and we need **to compile** which leads to higher *maintenance* activities.

In order to avoid recompiling a *jdbc* application we must develop a flexible *jdbc* application with the help of **resource bundle file** or **properties file**.

A *resource bundle file* or *properties file* is one which contains the data in the form of (key, value) pair.

For example:

```
<file name>.<rbf/prop>
```

NOTE: After creating resource bundle file that file must be stored *into* current working directory.

How to read the data from RESOURCE BUNDLE FILE:

1. In order to read the data from resource bundle file, open the resource bundle file in read mode with the help of *FileInputStream* class.

For example:

```
FileInputStream fis=new FileInputStream ("db.prop");
```

2. Since files does not support to read the data separately in the form of (key, value). Hence, it is recommended to get the data of the file we must create an object of a predefined class called *java.util.Properties*

For example:

```
Properties p=new Properties ();
```

3. In order to link 'fis' and 'p' objects we must use the following method:

```
java.util.Properties  
    ↓  
public void load (FileInputStream);
```

For example:

```
p.load (fis);
```

4. Obtain the property value by passing property name by using the following method:

```
public Object get (String);  
public Object getProperty (String);
```

Here, String represents property name or key name.

For example:

```
String dname= (String) p.get ("Dname");  
String url= (String) p.get ("URL");  
String username= (String) p.get ("Uname");  
String password= (String) p.get ("Pwd");  
String table= (String) p.get ("Tablename");
```

Here, Dname, URL, Uname, Pwd and Tablename are the property names present in resource bundle file. dname, url, username, password and table are the property values present in **resource bundle file**.

Write a java program which illustrates the concept of resource bundle file or how to develop a flexible *jdbc* application along with its metadata?

Answer:

rbfdb.prop:

```
Dname=oracle.jdbc.driver.OracleDriver  
URL=jdbc:oracle:thin:@127.0.0.1:1521:oradsn  
Uname=scott
```

```
Pwd=tiger
Tablename=student
```

RBFCConcept:

```
import java.sql.*;
import java.io.*;
import java.util.*;
class RBFCConcept
{
    public static void main (String [] args)
    {
        try
        {
            FileInputStream fis=new FileInputStream ("rbfdb.prop");
            Properties p=new Properties ();
            p.load (fis);
            String dname= (String) p.get ("Dname");
            String url= (String) p.get ("URL");
            String username= (String) p.get ("Uname");
            String password= (String) p.get ("Pwd");
            String tablename= (String) p.get ("Tablename");

            // loading drivers and obtaining connection
            Class.forName (dname);
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection (url, username, password);
            System.out.println ("CONNECTION CREATED...");

            // executing query
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from"+tablename);
            ResultSetMetaData rsmd=rs.getMetaData ();

            // printing column names
            System.out.println ("=====");
            for (int i=1; i<=rsmd.getColumnCount (); i++)
            {
                System.out.print (rsmd.getColumnName (i)+" ");
            }
            System.out.println ("");
            System.out.println ("=====");

            // printing the data
            while (rs.next ())
            {
                for (int j=1; j<=rsmd.getColumnCount (); j++)
                {
                    System.out.print (rs.getString (j)+" ");
                }
            }
            con.close ();
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
}
```

```
    } // main  
}; // RSFConcept
```

Scrollable *ResultSet*'s and Updatable *ResultSet*'s:

Whenever we create an object of *ResultSet* by default, it allows us to retrieve in **forward direction** only and we cannot perform any modifications on *ResultSet* object. Therefore, by default the *ResultSet* object is non-scrollable and non-updatable *ResultSet*.

Day - 17:

Scrollable *ResultSet*:

A scrollable *ResultSet* is one which allows us to retrieve the data in forward direction as well as backward direction but no updations are allowed. In order to make the non-scrollable *ResultSet* as scrollable *ResultSet* as scrollable *ResultSet* we must use the following *createStatement* which is present in *Connection* interface.

```
java.sql.Connection  
    |  
    v  
public Statement createStatement (int Type, int Mode);
```

Type represents type of scrollability and Mode represents either read only or updatable. The value of Type and value of Mode are present in *ResultSet* interface as constant data members and they are:

int Type	int Mode
TYPE_FORWARD_ONLY → 1	CONCUR_READ_ONLY → 3
TYPE_SCROLL_INSENSITIVE → 2	

For example:

```
Statement st=con.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);  
ResultSet rs=st.executeQuery ("select * from student");
```

Whenever we create a *ResultSet* object, by default, constant-1 as a Type and constant-2 as a Mode will be assigned.

The following methods which are available in *ResultSet* interface which allows us to retrieve the data either in forward direction or in backward direction or in random retrieval:

```
public boolean next (); → 1  
public void beforeFirst (); → 2  
public boolean isFirst (); → 3  
public void first (); → 4  
public boolean isBeforeFirst (); → 5  
public boolean previous (); → 6  
public void afterLast (); → 7  
public boolean isLast (); → 8  
public void last (); → 9  
public boolean isAfterLast (); → 10  
public void absolute (int); → 11  
public void relative (int); → 12
```

- Method-1 returns true when rs contains next record otherwise false.

- Method-2 is used for making the *ResultSet* object to *point* to just before the first record (it is by default).
- Method-3 returns true when rs is *pointing* to first record otherwise false.
- Method-4 is used to *point* the *ResultSet* object to first record.
- Method-5 returns true when rs *pointing* to before first record otherwise false.
- Method-6 returns true when rs contains previous record otherwise false.
- Method-7 is used for making the *ResultSet* object to *point* to just after the last record.
- Method-8 returns true when rs is *pointing* to last record otherwise false.
- Method-9 is used to *point* the *ResultSet* object to last record.
- Method-10 returns true when rs is *pointing* after last record otherwise false.
- Method-11 is used for moving the *ResultSet* object to a particular record either in forward direction or in backward direction with respect to first record and last record respectively. If *int* value is positive, rs move in forward direction to that with respect to first record. If *int* value is negative, rs move in backward direction to that with respect to last record.
- Method-12 is used for moving rs to that record either in forward direction or in backward direction with respect to current record.

Write a java program which illustrates the concept of scrollable *ResultSet*?

Answer:

```
import java.sql.*;
class ScrollResultSet
{
    public static void main (String [] args)
    {
        try
        {
            Class.forName ("Sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");
            System.out.println ("CONNECTION ESTABLISHED...");
            Statement st=con.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.
CONCUR_READ_ONLY);

            ResultSet rs=st.executeQuery ("select * from emp");
            System.out.println ("RECORDS IN THE TABLE...");
            while (rs.next ())
            {
                System.out.println (rs.getInt (1)+"    "+rs.getString (2));
            }
            rs.first ();
            System.out.println ("FIRST RECORD...");
            System.out.println (rs.getInt (1)+"    "+rs.getString (2));
            rs.absolute (3);
            System.out.println ("THIRD RECORD...");
            System.out.println (rs.getInt (1)+"    "+rs.getString (2));
            rs.last ();
            System.out.println ("LAST RECORD...");
            System.out.println (rs.getInt (1)+"    "+rs.getString (2));
            rs.previous ();
            rs.relative (-1);
            System.out.println ("FIRST RECORD...");
            System.out.println (rs.getInt (1)+"    "+rs.getString (2));
```

```
        con.close ();
    }
    catch (Exception e)
    {
        System.out.println (e);
    }
} // main
}; // ScrollResultSet
```

Day - 18:

Updatable ResultSet:

Whenever we create a *ResultSet* object which never allows us to update the database through *ResultSet* object and it allows retrieving the data only in forward direction. Such type of *ResultSet* is known as non-updatable and non-scrollable *ResultSet*.

In order to make the *ResultSet* object as updatable and scrollable we must use the following constants which are present in *ResultSet* interface.

int Type
TYPE_SCROLL_SENSITIVE

int Mode
CONCUR_UPDATABLE

The above two constants must be specified while we are creating Statement object by using the following method:

```
java.sql.Connection  
    ↓  
public Statement createStatement (int Type, int Mode);
```

For example:

```
Statement st=con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE) ;
```

On *ResultSet* we can perform the following three operations, they are **inserting** a record, **deleting** a record and **updating** a record.

Steps for INSERTING a record through ResultSet object:

1. Decide at which position we are inserting a record by calling absolute method.

For example:

```
rs.absolute (3);
```

2. Since we are inserting a record we must use the following method to make the *ResultSet* object to hold the record.

```
java.sql.ResultSet  
    ↓  
public void moveToInsertRow();
```

For example:

```
rs.moveToInsertRow ();
```

3. Update all columns of the database or provide the values to all columns of database by using the following generalized method which is present in *ResultSet* interface.

```
java.sql.ResultSet  
    ↓  
public void updateXXX(int colno, XXX value);
```

For example:

```
rs.updateInt (1, 5);  
rs.updateString (2, "abc");  
rs.updateInt (3, 80);
```

4. Upto step-3 the data is inserted in *ResultSet* object and whose data must be inserted in the database permanently by calling the following method:

```
java.sql.ResultSet  
    ↓  
public void insertRow();
```

It throws an exception called *SQLException*.

For example:

```
rs.insertRow ();
```

Steps for DELETING a record through *ResultSet* object:

1. Decide which record you want to delete.

For example:

```
rs.absolute (3); // rs pointing to 3rd record & marked for deletion
```

2. To delete the record permanently from the database we must call the following method which is present in *ResultSet* interface.

```
java.sql.ResultSet  
    ↓  
public void deleteRow();
```

For example:

```
rs.deleteRow ();
```

Steps for UPDATING a record through *ResultSet* object:

1. Decide which record to update.

For example:

```
rs.absolute (2);
```

2. Decide which columns to be updated.

For example:

```
rs.updateString (2, "pqr");  
rs.updateInt (3, 91);
```

3. Using step-2 we can modify the content of *ResultSet* object and the content of *ResultSet* object must be updated to the database permanently by calling the following method which is present in *ResultSet* interface.

```
java.sql.ResultSet  
    ↓  
public void updateRow();
```

For example:

```
rs.updateRow ();
```

Write a java program which illustrates the concept of **updatable ResultSet?**

Answer:

```
import java.sql.*;  
class UpdateResultSet  
{  
    public static void main (String [] args)  
    {  
        try  
        {  
            Class.forName ("Sun.jdbc.odbc.JdbcOdbcDriver");  
            System.out.println ("DRIVERS LOADED...");  
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");  
            System.out.println ("CONNECTION ESTABLISHED...");  
            Statement st=con.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.  
CONCUR_UPDATABLE);  
            ResultSet rs=st.executeQuery ("select * from emp1");  
            rs.next ();  
            rs.updateInt (2,8000);  
            rs.updateRow ();  
            System.out.println ("1 ROW UPDATED...");  
            rs.moveToInsertRow ();  
            rs.updateInt (1, 104);  
            rs.updateInt (2, 2000);  
            rs.insertRow ();  
            System.out.println ("1 ROW INSERTED...");  
            rs.absolute (2);  
            rs.deleteRow ();  
            System.out.println ("1 ROW DELETED...");  
            con.close ();  
        }  
        catch (Exception e)  
        {  
            e.printStackTrace ();  
        }  
    }  
    }  
} // main  
}; // UpdateResultSet
```

NOTE:

The scrollability and updatability of a *ResultSet* depends on the development of the driver of the driver vendors. *OracleDriver* and *JdbcOdbcDriver* will support the concept of scrollability and updatability of a *ResultSet* but there may be some drivers which are available in the industry which are not supporting the concept of scrollability and updatability.

BATCH PROCESSING

In traditional *jdbc* programming, to perform any transaction (insert, update and delete) we must send a separate request to database. If there is 'n' number of transactions then we must make 'n' number of request to the database and finally it leads to poor performance.

Disadvantages of NON-BATCH PROCESSING applications:

1. There is a possibility of leading the database result in inconsistent in the case of *interrelated* transactions.
2. Number of Network Round Trips or to and fro calls are more between frontend and backend application.

To avoid the above disadvantages we must use batch processing.

Batch processing is the process of grouping 'n' number of *interrelated* transactions in a single unit and processing at a same time.

Day - 19:

Advantages of BATCH PROCESSING:

1. Batch processing always leads consistency of the database.
2. Number of network round trips or to and fro calls will be reduced.

Steps for developing BATCH PROCESSING application:

1. Every batch processing application must contain only **DML** (insert, delete and update) statements.
2. Set the auto commit as **false**. Since, in *jdbc* environment by default auto commit is **true**. In order to make auto commit as false we must use the following method:

```
java.sql.Connection ──┐  
                        │  
                        ▼  
public void setAutoCommit (boolean);
```

For example:

```
con.setAutoCommit (false);
```

3. Prepare set of SQL DML statements and add to batch.

```
java.sql.Statement ──┐  
                      │  
                      ▼  
public void addBatch (String);
```

For example:

```
st.addBatch ("insert into dept values (10,"abc","hyd")");
```

4. Execute batch of DML statements by using the following method:

```
java.sql.Statement ──┐  
                      │  
                      ▼  
public int [] executeBatch ();
```

For example:

```
int res []=st.executeBatch ();
```

This method returns individual values of DML statements.

5. After completion of batch of statements, commit the database by using the following method:

```
java.sql.Connection  
└─  
public void commit ();
```

For example:

```
con.commit ();
```

6. If a single batch statement is not executing then we must rollback the database if required to its original state by using the following method:

```
java.sql.Connection  
└─  
public void rollback ();
```

For example:

```
con.rollback ();
```

Write a java program which illustrates the concept of Batch processing?

Answer:

```
import java.sql.*;  
class BatchProConcept  
{  
    public static void main (String [] args) throws Exception  
    {  
        Class.forName ("Sun.jdbc.odbc.JdbcOdbcDriver");  
        System.out.println ("DRIVERS LOADED...");  
        Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");  
        System.out.println ("CONNECTION ESTABLISHED...");  
        con.setAutoCommit (false);  
        Statement st=con.createStatement ();  
        st.addBatch ("insert into student values (3, 'j2ee')");  
        st.addBatch ("delete from student where sno=1");  
        st.addBatch ("update student set sname='java' where sno=2");  
        int res []=st.executeBatch ();  
        for (int i=0; i<res.length; i++)  
        {  
            System.out.println ("NUMBER OF ROWS EFFECTED : "+res [i]);  
        }  
        con.commit ();  
        con.rollback ();  
        con.close ();  
    }  
} // main  
}; // BatchProConcept
```

With batch processing we can obtain effective performance to the *jdbc* applications by executing group of SQL DML statements.

Write a java program to create a table through frontend application?

Answer:

```
import java.sql.*;  
class CreateTable
```

```
{
    public static void main (String [] args)
    {
        try
        {
            Class.forName ("Sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println ("DRIVERS LOADED...");
            Connection con=DriverManager.getConnection ("jdbc:odbc:oradsn","scott","tiger");
            System.out.println ("CONNECTION ESTABLISHED...");
            Statement st=con.createStatement ();
            int i=st.executeUpdate ("create table kalyan (eno number (4), ename varchar2 (15))");
            System.out.println ("TABLE CREATED...");
            con.close ();
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
} // main
}; // CreateTable
```

Dealing with DATE

- In order to deal with java date or frontend application date we should use a class called `java.util.Date`
- In order to deal with database date or backend application date we should use a class called `java.sql.Date`
- `java.sql.Date` is the subclass of `java.util.Date`

Converting java.util.Date into java.sql.Date:

When we are dealing with `frontend application` we must always take an object of `java.util.Date` for representing date and time information but when we are dealing with `database` date's we must take an object of `java.sql.Date`

In order to convert `java.util.Date` into `java.sql.Date` we have two ways:

First way:

Read the date in string format

For example:

```
System.out.println ("ENTER THE DATE IN DD-MM-YYYY");
String d1=dis.readLine ();
```

To convert string date into `java.sql.Date` we have the following method:

```
java.sql.Date
      |
      v
public static java.sql.Date valueOf (String);
```

For example:

```
java.sql.Date sd=java.sql.Date.valueOf (d1);
```

Second way:

In order to convert string date into `java.util.Date` we must use the following class:

```
SimpleDateFormat  
↓  
SimpleDateFormat (String Date_format);  
  
SimpleDateFormat  
↓  
public java.util.Date parse (String Sdate);
```

For example:

```
SimpleDateFormat sdf=new SimpleDateFormat ("DD-MM-YYYY");  
java.util.Date ud=sdf.parse (d1);
```

To convert java.util.Date into java.sql.Date we must use the following statement:

For example:

```
java.sql.Date sd=new java.util.Date (ud.getTime ());
```

Day - 20:**SERVLETS**

Any company want to develop the website that can be developed in two ways, they are **static website** and **dynamic website**.

- A *static website* is one where there is no *interaction* from the end user. To develop static website we can use the markup languages like **HTML, DHMTL, XML, JavaScript etc.**
- A *dynamic website* is one in which there exist end user *interaction*. To develop dynamic websites, in industry we have lot of technologies such as CGI (**Common Gateway Interface**), java, dot net, etc.

In the recent years SUN micro systems has developed a technology called **Servlets** to develop the dynamic websites and also for developing distributed applications.

A distributed application is one which always runs in the **context of browser** or **www**. The result of distributed application is always sharable across the globe. To develop distributed application one must follow the following:

Client-Server architecture:

- 2-tier architecture (Client program, Database program).
- 3-tier or MVC (**Model** [Database] **View** [JSP] **Controller** [Servlets]) architecture (Client program, Server program, Database program).
- n-tier architecture (Client program, Firewall program, Server program, Database program).

To exchange the data between client and server we use a protocol caller **http** which is a part of TCP/IP.

- A *client* is the program which always **makes a request to get the service from server**.
- A *server* is the program which always **receives the request, process the request and gives response to 'n' number of clients concurrently**.

A server is the third party software developed by third party vendors according to SUN micro systems specification. All servers in the industry are developed in java language only. The basic purpose of using server is that to get concurrent access to a server side program.

According to industry scenario, we have two types of servers; they are **web server** and **application server**.

WEB SERVER

1. A web server is one which always supports *http* protocol only.
2. Web server does not contain enough security to prevent unauthorized users.
3. Web server is not able to provide enough services to develop effective server side program.
4. For examples Tomcat server, web logic server, etc.

APPLICATION SERVER

1. Any protocol can be supported.
2. An application server always provides 100% security to the server side program.
3. An application server provides effective services to develop server side program.
4. For examples web logic server, web sphere server, pramathi server, etc.

NOTE: Web logic server acts as both web server and as well as application server.

In the *initial* days of server side programming there is a concept called CGI and this was implemented in the languages called C and PERL. Because of this approach CGI has the following disadvantages.

1. Platform dependency.
2. Not enough security is provided.
3. Having lack of performance. Since, for each and every request a new and separate **process** is creating (for example, if we make hundreds of requests, in the server side hundreds of new and separate processes will be created)

To avoid the above problems SUN micro system has released a technology called **Servlets**.

A *servlet* is a simple platform independent, architectural neutral server independent java program which extends the functionality of either **web server** or **application server** by running in the **context of www**.

Advantages of SERVLETS over CGI:

1. Servlets are always platform independent.
2. Servlets provides 100% security.
3. Irrespective of number of requests, a **single process** will be created at server side. Hence, Servlets are known as **single instance multiple thread technology**.

Day - 21:

Servlets is the standard specification released by SUN micro systems and it is implemented by various server vendors such as BEA corporation (Web logic server), Apache Jakarta (Tomcat server).

In order to run any servlet one must have either application server or web server. In order to deal with servlet programming we must import the following packages:

```
javax.servlet.*;
```

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

Servlet Hierarchy:

- In the above hierarchy chart Servlet is an **interface** which contains three life cycle methods without definition.
- GenericServlet is an abstract class which implements Servlet **interface** for defining life cycle methods i.e., life cycle methods are defined in GenericServlet with **null** body.
- Using GenericServlet class we can develop **protocol independent applications**.
- HttpServlet is also an abstract class which extends GenericServlet and by using this class we can develop **protocol dependent applications**.
- To develop our own servlet we must choose a class that must extends either GenericServlet or HttpServlet.

LIFE CYCLE METHODS of servlets:

In servlets we have three life cycle methods, they are

```
public void init ();  
public void service (ServletRequest req, ServletResponse res);  
public void destroy ();
```

public void init ():

Whenever client makes a request to a servlet, the server will receive the request and it automatically calls *init ()* method i.e., *init ()* method will be called **only one** time by the server whenever we make first request.

In this method, we write the block of statements which will perform **one time operations**, such as, opening the file, database connection, *initialization* of parameters, etc.

public void service (ServletRequest, ServletResponse):

After calling *init ()* method, *service ()* method will be called when we make first request from second request to further subsequent requests, server will call only *service* method. Therefore, *service ()* method will be called each and every time as and when we make a request.

In *service ()* method we write the block of statements which will perform repeated operations such as retrieving data from database, retrieving data from file, modifications of parameters data, etc. Hence, in *service ()* method we always write business logic.

Whenever control comes to *service ()* method the server will create two objects of **ServletRequest** and **ServletResponse** *interfaces*.

Object of *ServletRequest* contains the data which is passed by client. After processing client data, the resultant data must be kept in an object of *ServletResponse*.

An object of *ServletRequest* and *ServletResponse* must be used always within the scope of *service ()* method only i.e., we cannot use in *init ()* method and *destroy ()* method.

Once the *service ()* method is completed an object of *ServletRequest* and an object of *ServletResponse* will be destroyed.

public void destroy ():

The *destroy ()* method will be called by the server in **two situations**; they are **when the server is closed** and **when the servlet is removed from server context**. In this method we write the block of statements which are obtained in *init ()* method.

NOTE: Life cycle methods are those which will be called by the server at various times to perform various operations.

Write a servlet which displays a message “I LOVE MY MOM”?

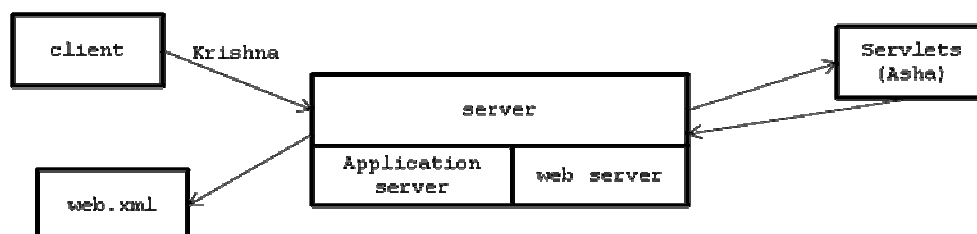
Answer:

```
import javax.servlet.*;
import java.io.*;
public class First extends GenericServlet
{
    public First ()
    {
        System.out.println ("I AM FROM DEFAULT CONSTRUCTOR...");
    }
    public void init ()
    {
        System.out.println ("I AM FROM init METHOD...");
    }
    public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException
    {
        System.out.println ("I LOVE MY MOM...");
        System.out.println ("I AM FROM service METHOD...");
    }
    public void destroy ()
    {
        System.out.println ("I AM FROM destroy METHOD...");
    }
};
```

Day - 22:

web.xml:

1. Whenever client makes a request to a servlet that request is received by server and server goes to a predefined file called web.xml for the details about a servlet.
2. **web.xml** file always gives the details of the servlets which are available in the server.
3. If the server is not able to find the requested servlet by the client then server generates an error (resource not found) [A *resource* is a program which resides in server].
4. If the requested servlet is available in web.xml then server will go to the servlet, executes the servlet and gives response back to client.



Every **web.xml** will contain the following entries:

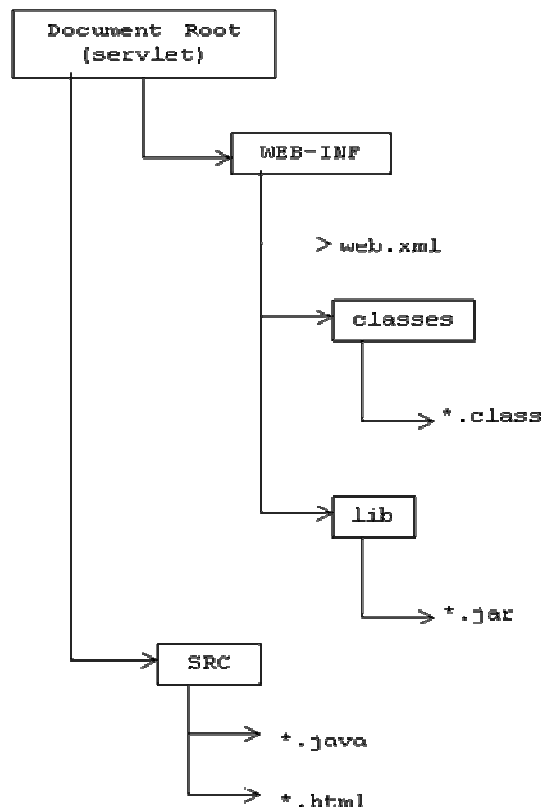
```
<web-app>
    <servlet>
        <servlet-name>Asha</servlet-name>
        <servlet-class>First</servlet-class>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>Asha</servlet-name>
    <url-pattern>Krishna</url-pattern>
</servlet-mapping>
</web-app>
```

- While server is executing *web.xml* control goes to `<url-pattern>` of `<servlet-mapping>` tag. If the requested url and `<url-name>` *web.xml* is same then control goes to `<servlet-class>` tag of `<servlet>` and takes the `<servlet-name>` and executes.
- If the `<url-pattern>` is not matching, server generates an error.

How to execute the servlets:

In order to execute a servlet we must follow the following directory structure:



Day - 23:

Steps for DEVELOPING a servlet:

1. Import `javax.servlet.*`, `javax.servlet.http.*` and other packages if required.
2. Choose user defined class.
3. Whichever class we have chosen in step-2 must extend either `GenericServlet` or `HttpServlet`.
4. Override the life cycle methods if required.

FLOW OF EXECUTION in a servlet:

1. Client makes a request. The general form of a request is [http://\(IP address or DNS \[Domain Naming Service\] name of the machine where server is installed\) : \(port number of the server\) / \(Document root\) : \(Resource name\).](http://(IP address or DNS [Domain Naming Service] name of the machine where server is installed) : (port number of the server) / (Document root) : (Resource name).)

For example:

<http://localhost:7001/DateSer/suman>

2. Server receives the request.
3. Server will scan *web.xml* (contains **declarative details**) if the requested resource is not available in *web.xml* server generates an error called resource not available otherwise server goes to a servlet.
4. Server will call the servlet for executing.
5. Servlet will execute in the **context of server**.
6. While server is executing a servlet, server loads an object of servlet class only once (by calling default constructor).
7. After loading the servlet, the servlet will call *init ()* method only once to perform one time operations.
8. After completion of *init ()* method, *service ()* method will be called each and every time. As long as we make number of requests only *service ()* method will be called to provide business logic.
9. Servlet will call *destroy ()* method either in the case of servlet is removed or in the case of server is closed.

HOW TO EXECUTE a servlet:

1. Prepare a directory structure.
2. Write a servlet program save it *into* either document root or document root\SRC.
3. Compile a servlet by setting a classpath.

For Tomcat:

Set classpath=

4. Copy *.class file *into* document root/WEB-INF/classes folder and write *web.xml* file.
5. Start the server and copy document root *into*:
6. Open the browser and pass a request or url

Write a servlet which displays current system date and time?

Answer:

Servlet program: (Since, it's a package to compile use *javac -d . DateServ.java*)

```
package ds;
import javax.servlet.*;
import java.io.*;
import java.util.*;
public class DateServ extends GenericServlet
{
    public DateServ ()
    {
        System.out.println ("SERVLET LOADED...");
    }
    public void init ()
    {
        System.out.println ("I AM FROM init METHOD...");
    }
    public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        Date d=new Date ();
        String s=d.toString ();
```



```
        pw.println("<h1> WELCOME TO SERVLETS <h1>");
        pw.println("<h2> CURRENT DATE & TIME IS : "+s+"<h2>");
    }
    public void destroy ()
    {
        System.out.println ("I AM FROM destroy METHOD...");
    }
};
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>kalyan</servlet-name>
        <servlet-class>ds.DateServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>kalyan</servlet-name>
        <url-pattern>/suman</url-pattern>
    </servlet-mapping>
</web-app>
```

Day - 24:**HttpServlet:**

- HttpServlet is the sub-class of GenericServlet.
- HttpServlet contains all the life cycle methods of GenericServlet and the service () method of GenericServlet is further divided *into* the following two methods, they are
public void doGet (HttpServletRequest, HttpServletResponse) throws ServletException, IOException
public void doPost (HttpServletRequest, HttpServletResponse) throws ServletException, IOException
- Whenever client makes a request, the **servlet container** (server) will call service () method, the service () method depends on type of the method used by the client application.
- If client method is *get* then service () method will call doGet () method and doGet () method *internally* creates the objects of HttpServletRequest and HttpServletResponse. Once doGet () method is completed its execution, the above two objects will be destroyed.

LIMITATIONS of get method:

1. Whatever data we sent from client by using *get* method, the client data will be populated or appended as a part of URL.

For example:

http://localhost:7001/servlet/DDservlet?uname=scott&pwd=tiger

2. Large amount of data cannot be transmitted from client side to server side.
- When we use *post* method to send client data, that data will be send as a part of method body and *internally* the service () method will called doPost () method by creating the objects of HttpServletRequest and HttpServletResponse.

ADVANTAGES of post method:

1. Security is achieved for client data.
2. We can send large amount of data from client to server.

- *HttpServletRequest* extends *ServletRequest* and *HttpServletResponse* extends *ServletResponse*.
- *HttpServlet*, *HttpServletRequest* and *HttpServletResponse* belong to a package called *javax.servlet.http.**
- The request which we make from the client side that requests are known as http requests where as the responses which are given by a servlet are known as http responses.

NOTE: All real world applications always extends *HttpServlet* only and it is always recommended to overwrite either *doGet ()* method or *doPost ()* method.

Associated with servlet we have three names which are specified in *web.xml*, they are **public URL name** (known to everybody), **deployer URL name** or **dummy name** (known to that person who is deploying) and **secret** or **internal URL name** (known to servlet container or server).

- The purpose of `<servlet-mapping>` is that it maps **public URL name** to **deployer URL name**.
- The purpose of `<servlet>` is that it maps **deployer URL name** to actual **Servlet class name**.

Write a servlet which retrieves the data from database?

Answer:

Servlet program:

```
package ddrs;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
public class RetrieveDataBaseServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res)
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        try
        {
            DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
            Connection con=DriverManager.getConnection ("oracle:jdbc:thin:@localhost:1521:
Hanuman", "scott", "tiger");
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from emp");
            while (rs.next ())
            {
                pw.println (rs.getString (1)+" "+rs.getString (2));
            }
        }
        catch (Exception e)
        {
            res.sendError (504, "PROBLEM IN SERVLET...");
        }
    }
};
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>Babu</servlet-name>
        <servlet-class>ddrs.RetrieveDataBaseServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Babu</servlet-name>
        <url-pattern>Dinesh</url-pattern>
    </servlet-mapping>
</web-app>
```

Day - 25:

In the above program we are making use of 'n' number of servlet classes, OracleDriver class; it is required to set the classpath for servlet-api.jar and *classes111.jar*

For weblogic:

```
set classpath=F:\bea\weblogic81\server\lib\weblogic.jar;F:\oracle\ora92\jdbc\
lib\classes111.jar;
```

For Tomcat:

```
set classpath=F:\Program Files\Apache Software Foundation\Tomcat 5.5\common\
lib\servlet-api.jar;F:\oracle\ora92\jdbc\lib\classes111.jar;
```

When we run the above program on weblogic, it is not necessary to copy *classes111.jar* into lib folder of document root. Since, the weblogic server itself contains an existing jar file called *ojdbc14.jar* to deal with OracleDriver.

How to generate a war file:

A war file is the compressed form of 'n' number of .class files, web.xml, *.html files and the jar files available in lib folder.

Syntax:

```
jar cfv (name of the war file) WEB-INF [*.html] [*.jsp]
```

Here, in *cfv*, 'c' represents create, 'f' represents file and 'v' represents verbose (used to compress)

Copy the war file from the current directory and paste it *into* applications folder of weblogic or webapps folder of Tomcat.

ServletConfig (one per SERVLET):

- *ServletConfig* is an *interface* which is present in *javax.servlet.** package.
- The purpose of *ServletConfig* is to pass some *initial* parameter values, technical information (driver name, database name, data source name, etc.) to a servlet.
- An object of *ServletConfig* will be created one per servlet.
- An object of *ServletConfig* will be created by the server at the time of executing public void *init* (*ServletConfig*) method.
- An object of *ServletConfig* cannot be accessed in the default constructor of a Servlet class. Since, at the time of executing default constructor *ServletConfig* object does not exist.

- By default *ServletConfig* object can be accessed with in *init ()* method only but not in *doGet* and *doPost*. In order to use, in the entire servlet preserve the reference of *ServletConfig* into another variable and declare this variable into a Servlet class as a data member of *ServletConfig*.

For example:

```
class x extends HttpServlet
```

Day - 26:

- When we want to give some global data to a servlet we must obtain an object of *ServletConfig*.
- web.xml entries for *ServletConfig*

```
<servlet>
.....
<init-param>
    <param-name>Name of the parameter</param-name>
    <param-value>Value of the parameter</param-value>
</init-param>
.....
</servlet>
```

For example:

```
<servlet>
    <servlet-name>abc</servlet-name>
    <servlet-class>serv1</servlet-class>
    <init-param>
        <param-name>v1</param-name>
        <param-value>10</param-value>
    </init-param>
    <init-param>
        <param-name>v2</param-name>
        <param-value>20</param-value>
    </init-param>
</servlet>
```

The data which is available in *ServletConfig* object is in the form of (key, value)

OBTAINING an object of *ServletConfig*:

An object of *ServletConfig* can be obtained in two ways, they are **by calling *getServletConfig ()* method** and **by calling *init (ServletConfig)***.

By calling *getServletConfig ()* method:

getServletConfig () is the method which is available in *javax.servlet.Servlet* interface. This method is further inherited and defined into a class called *javax.servlet.GenericServlet* and that method is further inherited into another predefined class called *javax.servlet.http.HttpServlet* and it can be inherited into our own servlet class.

For example:

```
public class serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
```

```

.....
.....
ServletConfig config=this.getServletConfig ();
.....
.....
}
};

```

In the above example an object config contains (key, value) pair data of web.xml file which are written under `<init-param>` tag of `<servlet>` tag.

By calling *init* (ServletConfig):

For example:

```

public class serv2 extends HttpServlet
{
    ServletConfig sc;
    public void init (ServletConfig sc)
    {
        Super.init (sc); // used for calling init (ServletConfig) method of HttpServlet
        this.sc=sc; // ServletConfig object sc is referenced
    }
    .....
    .....
};

```

RETRIEVING DATA from ServletConfig interface object:

In order to get the data from *ServletConfig* interface object we must use the following methods:

```

public String getInitParameter (String); → 1
public Enumeration getInitParameterNames (); → 2

```

Method-1 is used for obtaining the parameter value by passing parameter name.

Parameter name ←	key	value	Parameter value →
	v1	10	
	v2	20	
	v3	30	

```

String val1=config.getInitParameter ("v1");
String val2=config.getInitParameter ("v2");
String val3=config.getInitParameter ("v3");

```

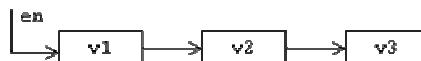
Method-2 is used for obtaining all parameter names and their corresponding parameter values.

For example:

```

Enumeration en=config.getInitParameterNames ();

```



```

while (en.hasMoreElements ())
{
    Object obj=en.nextElement ();
    String pname= (String) obj;
    String pvalue=config.getInitParameter (pname);
    out.println (pvalue+" is the value of "+pname);
}

```

```
}
```

Day - 27:**Serv1.java:**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletConfig config=getServletConfig ();
        String val1=config.getInitParameter ("v1");
        String val2=config.getInitParameter ("v2");
        String val3=config.getInitParameter ("v3");
        String val4=config.getInitParameter ("v4");
        pw.println ("<h3> Value of v1 is "+val1+"</h3>");
        pw.println ("<h3> Value of v2 is "+val2+"</h3>");
        pw.println ("<h3> Value of v3 is "+val3+"</h3>");
        pw.println ("<h3> Value of v4 is "+val4+"</h3>");
    }
};
```

Serv2.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletConfig config=getServletConfig ();
        Enumeration en=config.getInitParameterNames ();
        while (en.hasMoreElements ())
        {
            Object obj=en.nextElement ();
            String pname= (String) obj;
            String pvalue=config.getInitParameter (pname);
            pw.println ("</h2>"+pvalue+" is the value of "+pname+"</h2>");
        }
    }
};
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>Serv1</servlet-class>
```

```
<init-param>
    <param-name>v1</param-name>
    <param-value>10</param-value>
</init-param>
<init-param>
    <param-name>v2</param-name>
    <param-value>20</param-value>
</init-param>
</servlet>
<servlet>
    <servlet-name>pqr</servlet-name>
    <servlet-class>Serv2</servlet-class>
    <init-param>
        <param-name>v3</param-name>
        <param-value>30</param-value>
    </init-param>
    <init-param>
        <param-name>v4</param-name>
        <param-value>40</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/firstserv</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>pqr</servlet-name>
    <url-pattern>/secondserv</url-pattern>
</servlet-mapping>
</web-app>
```

Develop a flexible servlet that should display the data of the database irrespective driver name, table name and dsn name?

Answer:

DbServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
public class DbServ extends HttpServlet
{
    ServletConfig sc=null;
    public void init (ServletConfig sc) throws ServletException
    {
        super.init (sc);
        this.sc=sc;
    }
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String dname=sc.getInitParameter ("dname");
        String url=sc.getInitParameter ("url");
        String tab=sc.getInitParameter ("tab");
        try
```

```
{
    Class.forName (dname);
    Connection con=DriverManager.getConnection (url,"scott","tiger");
    Statement st=con.createStatement ();
    ResultSet rs=st.executeQuery ("select * from "+tab);
    while (rs.next ())
    {
        pw.println ("<h2>" +rs.getString (1)+" "+rs.getString (2)+" "+rs.getString (3)+"</h2>");
    }
    con.close ();
}
catch (Exception e)
{
    res.sendError (503,"PROBLEM IN DATABASE...");
}
}
};
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>DbServ</servlet-class>
        <init-param>
            <param-name>dname</param-name>
            <param-value>oracle.jdbc.driver.OracleDriver ()</param-value>
        </init-param>
        <init-param>
            <param-name>url</param-name>
            <param-value>jdbc:oracle:thin:@localhost:1521:Hanuman</param-value>
        </init-param>
        <init-param>
            <param-name>tab</param-name>
            <param-value>emp</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/dbdata</url-pattern>
    </servlet-mapping>
</web-app>
```

Day - 28:**ServletContext (one per WEB APPLICATION):**

- *ServletContext* is an interface which is present in *javax.servlet.** package.
- Whenever we want to give a common data or global data to the group of servlets which belongs to same web application then we must create an object of *ServletContext* interface.
- An object of *ServletContext* will be created by servlet container (server) whenever we deploy into the server.
- In order to provide a common data to a group of servlets, we must write that data into web.xml file with the tag called `<context-param>...</context-param>`. This tag must be written with in `<web-app>...</web-app>` before `<servlet>`.

- xml entries related to *ServletContext* interface.

```
<web-app>
    <context-param>
        <param-name>Name of the param</param-name>
        <param-value>Value of the param</param-value>
    </context-param>
    <servlet>
        .....
    </servlet>
    <servlet-mapping>
        .....
    </servlet-mapping>
</web-app>
```

- Whatever the data we write with in `<context-param>...</context-param>` that data will be paste automatically in the object of *ServletContext* interface and this object contains the in the form of (key, value) pair. Here, *key* represents context parameter name and *value* represents context parameter value.
- The value of key must be always unique; if duplicate values are placed we get recent duplicate value for the key by overlapping previous values.

For example:

```
<web-app>
    <context-param>
        <param-name>driver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </context-param>
    <context-param>
        <param-name>url</param-name>
        <param-value>jdbc:oracle:thin:@localhost:1521:Hanuman</param-value>
    </context-param>
    <servlet>
        .....
    </servlet>
    <servlet-mapping>
        .....
    </servlet-mapping>
</web-app>
```

Number of ways to OBTAIN AN OBJECT of *ServletContext*:

In order to get an object of *ServletContext* we have two ways, they are **by calling `getServletContext ()` method directly** and **by making use of *ServletConfig* interface**.

By using `getServletContext ()` method:

`getServletContext ()` method is defined in *GenericServlet* and it is inherited into *HttpServlet* and it is further inherited into our own servlet class. Hence, we can call `getServletContext ()` method directly.

For example:

```
public class Serv1 extends HttpServlet
{
```

```
public doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    .....
    .....
    ServletContext ctx=this.getServletContext ();
    .....
    .....
}
};
```

By using ServletConfig interface:

In *ServletConfig* interface we have the following method which gives an object of *ServletContext*.

In order to call the above method first we must obtain an object of *ServletConfig* interface and later with that object we can call *getServletContext ()* method.

For example:

```
public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        .....
        .....
        ServletConfig config=this.getServletConfig ();
        ServletContext ctx=config.getServletContext ();
        .....
        .....
    }
};
```

Number of ways to RETRIEVE THE DATA from an OBJECT of ServletContext:

In *ServletContext* interface we have the following methods to retrieve the value of context parameter by passing context parameter name.

```
javax.servlet.ServletContext
    ↓
public String getInitParameter (String);
```

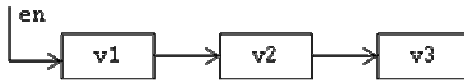
For example:

```
ServletContext ctx=getServletContext ();
String val1=ctx.getInitParameter ("v1");
String val2=ctx.getInitParameter ("v2");
```

```
javax.servlet.ServletContext
    ↓
public Enumeration getInitParameterNames ();
```

For example:

```
ServletContext ctx=getServletContext ();
Enumeration en=ctx.getInitParameterNames ();
While (en.hasMoreElements ())
{
    String cpn= (String) en.nextElement ();
    String cpv=ctx.getInitParameter (cpn);
    pw.println (cpv+" is the value of "+cpn);
}
```

**Differences between *ServletConfig* and *ServletContext* interfaces:*****ServletConfig***

1. An object of *ServletConfig* exists **one per servlet**.
2. An object of *ServletConfig* will be created when *init (ServletConfig)* method is executed.
3. *ServletConfig* object contains a **specific data** to a particular servlet.
4. The data to a servlet which related to *ServletConfig* object must be written in `<init-param>...</init-param>` with in `<servlet>...</servlet>` of web.xml
5. An object of *ServletConfig* will exists as long as a specific servlet is executing.

ServletContext

1. An object of *ServletContext* exists **one per web application**.
2. An object of *ServletContext* will be created when we deploy the web application in servlet container or servlet execution environment.
3. *ServletContext* object contains a common or global data to 'n' number of servlets and 'n' number of JSP's.
4. The common data or global data related to *ServletContext* must be written under `<context-param>...</context-param>` with in `<web-app>...</web-app>` and outside of `<servlet>...</servlet>` of web.xml
5. An object of *ServletContext* will exists until the entire web application completed its execution.

Day - 29:

Write a servlet which illustrate the concept of *ServletContext*?

Answer:**web.xml:**

```
<web-app>
  <context-param>
    <param-name>v1</param-name>
    <param-value>10</param-value>
  </context-param>
  <context-param>
    <param-name>v2</param-name>
    <param-value>20</param-value>
  </context-param>
  <servlet>
    <servlet-name>abc</servlet-name>
    <servlet-class>Servlet</servlet-class>
    <init-param>
      <param-name>v3</param-name>
      <param-value>30</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>pqr</servlet-name>
```

```
<servlet-class>Serv2</servlet-class>
<init-param>
    <param-name>v4</param-name>
    <param-value>40</param-value>
</init-param>
</servlet>
<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/firstserv</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>pqr</servlet-name>
    <url-pattern>/secondserv</url-pattern>
</servlet-mapping>
</web-app>
```

Serv1.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletConfig config=getServletConfig ();
        ServletContext ctx=getServletContext ();
        String val1=ctx.getInitParameter ("v1");
        String val2=ctx.getInitParameter ("v2");
        String val3=config.getInitParameter ("v3");
        String val4=config.getInitParameter ("v4");
        int sum=Integer.parseInt (val1)+Integer.parseInt (val2);
        pw.println ("<h3> Value of v1 is "+val1+"</h3>");
        pw.println ("<h3> Value of v2 is "+val2+"</h3>");
        pw.println ("<h3> Value of v3 is "+val3+"</h3>");
        pw.println ("<h3> Value of v4 is "+val4+"</h3>");
        pw.println ("<h2> Sum = "+sum+"</h2>");
    }
};
```

Serv2.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletContext ctx=getServletContext ();
        Enumeration en=ctx.getInitParameterNames ();
        while (en.hasMoreElements ())
```

```
        {  
            Object obj=en.nextElement ();  
            String cpname= (String) obj;  
            String cpvalue=ctx.getInitParameter (cpname);  
            pw.println ("</h2>"+cpvalue+" is the value of "+cpname+"</h2>");  
        }  
    }  
};
```

Handling CLIENT REQUEST FORM data:

- Whenever we want to send an input to a servlet that input must be passed through html form.
- An html form is nothing but various controls are inherited to develop an application.
- Every form will accept client data and it must send to a servlet which resides in server side.
- Since html is a static language which cannot validate the client data. Hence, in real time applications client data will be accepted with the help of html tags by developing form and every form must call a servlet.

Steps for DEVELOPING a FORM:

1. Use <form>...</form> tag.
2. To develop various controls through <input>...</input> tag and all <input> tag must be enclosed with in <form>...</form> tag.
3. Every form must call a servlet by using the following:

```
<form name="name of the form" action="either absolute or relative address" method="get or post">  
.....  
.....  
</form>
```

Write an html program to develop the following form:

Personal Information	
Enter ur name :	<input type="text"/>
Enter ur course :	<input type="text"/>
<input type="button" value="Send"/>	<input type="button" value="Clear"/>

Answer:

```
<html>  
<title>About Personal Data</title>  
<head><center><h3>Personal Information</h3></center></head>  
<body bgcolor="#D8BFD8">  
    <form name="persdata" action="./DataSer">  
        <center>  
            <table bgcolor="#D2B48C" border="1">  
                <tr>  
                    <th>Enter ur name : </th>  
                    <td><input type="submit" name="persdata_eurn" value=""></td>  
                </tr>  
                <tr>  
                    <th>Enter ur course : </th>
```

```

        <td><input type="text" name="persdata_eurc" value=""></td>
    </tr>
    <tr>
        <td align="center"><input type="button" name="persdata_send" value="Send"></td>
        <td align="center"><input type="reset" name="persdata_clear" value="Clear"></td>
    </tr>
</table>
</center>
</form>
</body>
</html>

```

Handling HTML DATA in SERVLET:

In order to handle or obtain the data html form in a servlet, we have the following method which is present in *HttpServletRequest*.

```

    javax.servlet.http.HttpServletRequest
                                |
                                v
    public String getParameter (String);
                                /      \
    Value of html                Html form field name
    form field name

```

For example:

```

String sno=req.getParameter ("sno");
String sname=req.getParameter ("sname");
String cname=req.getParameter ("cname");

```

req is an object of *HttpServletRequest*.

NOTE: An object of *HttpServletRequest* will be created automatically by servlet container and it contains client requested data.

Day - 30:

Write a servlet which accepts client request and display the client requested data on the browser?

Answer:

```

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

public class DataServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String name=req.getParameter ("persdata_eurn");
        String cname=req.getParameter ("persdata_eurc");
        pw.println ("<center><h3>HELLO..! Mr/Mrs. "+name+"</h3></center>");
        pw.println ("<center><h3>Your COURSE is "+cname+"</h3></center>");
    }

    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
}

```

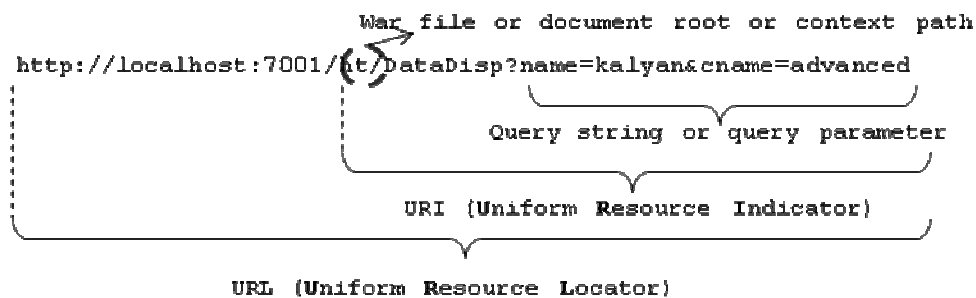
```
    }  
};
```

web.xml:

```
<web-app>  
    <servlet>  
        <servlet-name>abc</servlet-name>  
        <servlet-class>DataServ</servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>abc</servlet-name>  
        <url-pattern>/getdata</url-pattern>  
    </servlet-mapping>  
</web-app>
```

NOTE:

1. Whenever we are not entering the data in html form that data will be treated as empty space.
2. *Query string* represents the data which is passed by client to the servlet through html form. *URI* stands for Uniform Resource Indicator which gives the location of servlet where it is available. *URL* stands for Uniform Resource Locator which gives at which port number, in which server a particular JSP or servlet is running. *ht* represents a context path which can be either a document root or a war file.

**Day - 31:**

Write a servlet which accepts product details from html form and stores the product details into database?

Answer:**Product database:**

```
create table Product  
(  
    pid number (4),  
    pname varchar2 (15),  
    price number (6, 2)  
);
```

web.xml:

```
<web-app>  
    <servlet>  
        <servlet-name>abc</servlet-name>  
        <servlet-class>DatabaServ</servlet-class>  
    </servlet>
```

```
<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/getdata</url-pattern>
</servlet-mapping>
</web-app>
```

prodata.html:

```
<html>
<title>Product Data</title>
<head><center><h2>Product Information</h2></center></head>
<body bgcolor="#BDB76B">
    <center>
        <form name="prodata" action="./getdata" method="post">
            <table border="1" bgcolor="#E9967A">
                <tr>
                    <th>Enter product id : </th>
                    <td><input type="text" name="prodata_pid" value=""></td>
                </tr>
                <tr>
                    <th>Enter product name : </th>
                    <td><input type="text" name="prodata_name" value=""></td>
                </tr>
                <tr>
                    <th>Enter product price : </th>
                    <td><input type="text" name="prodata_price" value=""></td>
                </tr>
                <tr>
                    <table>
                        <tr>
                            <td align="center"><input type="submit" name="prodata_insert" value="Insert"></td>
                            <td align="center"><input type="reset" name="prodata_reset" value="Clear"></td>
                        </tr>
                    </table>
                </tr>
            </table>
        </form>
    </center>
</body>
</html>
```

DarabaServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class DatabaServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String pid1=req.getParameter ("prodata_pid");
        String pname=req.getParameter ("prodata_pname");
        String price1=req.getParameter ("prodata_price");
        int pid=Integer.parseInt (pid1);
        float price=Float.parseFloat (price1);
```



```

        try
        {
            DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
            Connection      con=DriverManager.getConnection      ("jdbc:oracle:thin:@localhost:1521:
Hanuman","scott","tiger");
            PreparedStatement ps=con.prepareStatement ("insert into Product values (?,?,:)");
            ps.setInt (1, pid);
            ps.setString (2, pname);
            ps.setFloat (3, price);
            int i=ps.executeUpdate ();
            pw.println ("<h4>"+i+" ROWS INSERTED...");
            con.close ();
        }
        catch (Exception e)
        {
            res.sendError (503, "PROBLEM IN DATABASE...");
        }
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};

```

VALIDATION of a form:

Develop the following html form and validate that form data by using a servlet?

Validations:

1. stno → must contain data and it should contain always *int* data.
2. sname → must contain data and no *special characters* are allowed.
3. smarks → must contain data and it should contain *float* data.

Answer:**Student database:**

```

create table Student
(
    stno number (3),
    stname varchar2 (15),
    stmarks number (5,2)
);

```

web.xml:

```

<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>ValidationServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>

```

```
        <url-pattern>/validation</url-pattern>
    </servlet-mapping>
</web-app>
```

validpro.html:

```
<html>
<title>Validation Project</title>
<head><center><h2>Student form validation</h2></center></head>
<body bgcolor="#FFE4C4">
    <center>
        <form name="validpro" action="./validation" method="post">
            <table border="1" bgcolor="A9A9A9">
                <tr>
                    <th>Enter student number : </th>
                    <td><input type="text" name="validpro_sno" value=""></td>
                </tr>
                <tr>
                    <th>Enter student name : </th>
                    <td><input type="text" name="validpro_sname" value=""></td>
                </tr>
                <tr>
                    <th>Enter student marks : </th>
                    <td><input type="text" name="validpro_smmarks" value=""></td>
                </tr>
                <tr>
                    <table>
                        <tr>
                            <td><input type="submit" name="validpro_insert" value="Insert"></td>
                            <td><input type="reset" name="validpro_clear" value="Clear"></td>
                        </tr>
                    </table>
                </tr>
            </table>
        </form>
    </center>
</body>
</html>
```

ValidationServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
import java.util.*;
public class ValidationServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        ArrayList al=new ArrayList ();
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String sno1=req.getParameter ("validpro_sno");
        String sname=req.getParameter ("validpro_sname");
        String smarks1=req.getParameter ("validpro_smmarks");
        int sno=0;
```

```
float smarks=0;
if ((sno1==null)|| (sno1.equals ("")))
{
    al.add ("PROVIDE STUDENT NUMBER...");
}
else
{
    try
    {
        sno=Integer.parseInt ("sno1");
    }
    catch (NumberFormatException nfe)
    {
        al.add ("PROVIDE int DATA IN STUDENT NUMBER...");
    }
}
if ((sname==null)|| (sname.equals ("")))
{
    al.add ("PROVIDE STUDENT NAME...");
}
if ((smarks1==null)|| (smarks1.equals ("")))
{
    al.add ("PROVIDE STUDENT MARKS...");
}
else
{
    try
    {
        smarks=Float.parseFloat ("smarks1");
    }
    catch (NumberFormatException nfe)
    {
        al.add ("PROVIDE float DATA IN STUDENT MARKS...");
    }
}
if (al.size ()!=0)
{
    pw.println (al);
}
else
{
    try
    {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
Hanuman","scott","tiger");

        PreparedStatement ps=con.prepareStatement ("insert into Student values (?,?,?)");
        ps.setInt (1, sno);
        ps.setString (2, sname);
        ps.setFloat (3, smarks);
        int i=ps.executeUpdate ();
        if (i>0)
        {
            pw.println ("RECORD INSERTED...");
        }
        else
    }
}
```

```
        {  
            pw.println ("RECORD NOT INSERTED...");  
        }  
        con.close ();  
    }  
    catch (Exception e)  
    {  
        res.sendError (503, "PROBLEM IN DATABASE...");  
    }  
}  
}  
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
{  
    doGet (req, res);  
}  
};
```

Day - 32:

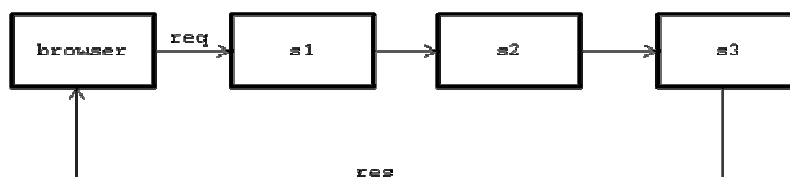
Servlet Chaining:

If a client request is processed by group of servlets, then that servlets are known as *servlet chaining* or if the group of servlets process a single client request then those servlets are known as *servlet chaining*.

In order to process a client request by many number of servlets then we have two models, they are **forward model** and **include model**.

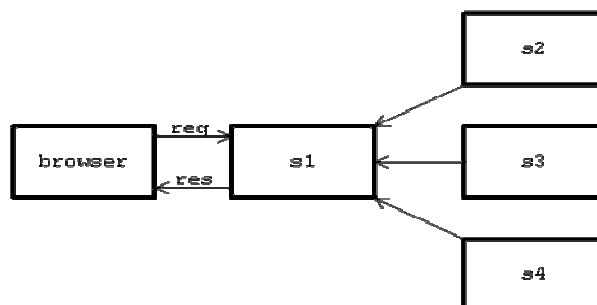
Forward model:

In this model when we forward a request to a group of servlets, finally we get the result of destination servlet as a response but not the result of intermediate servlets.



Include model:

If a single client request is passed to a servlet and that servlet makes use of other group of servlets to process a request by including the group of servlets into a single servlet.



In the above diagram client request goes to servlet s1 and s1 internally includes s2, s3 and s4 servlets and finally result of all these servlets given to the client by a source servlet s1.

NOTE: One servlet can include any number of servlets where as one servlet can forward to only one servlet at a time.

Steps for DEVELOPING Servlet Chaining:

1. Obtain an object of ServletContext by using in any of the following way:

```
ServletContext ctx1=getServletContext (); [GenericServlet method]
```

```
ServletContext ctx2=config.getServletContext (); [ServletConfig method]
```

```
ServletContext ctx3=req.getServletContext (); [HttpServletRequest method]
```

2. Obtain an object of RequestDispatcher. RequestDispatcher is an interface which is present in javax.servlet.* package and it is used for forwarding the request and response objects of source servlet to destination servlet or for including the destination servlet into source servlet. To obtain an object of RequestDispatcher, the ServletContext contains the following method:

```
javax.servlet.ServletContext  
↓  
public RequestDispatcher getRequestDispatcher (String);
```

```
RequestDispatcher rd=ctx.getRequestDispatcher ("/s2");
```

3. Use forward or include model to send the request and response objects. RequestDispatcher contains the following methods for forwarding or including the request and response objects.

```
javax.servlet.RequestDispatcher  
↓  
public void forward (ServletRequest, ServletResponse) throws ServletException, IOException
```

```
javax.servlet.RequestDispatcher  
↓  
public void include (ServletRequest, ServletResponse) throws ServletException, IOException
```

For example:

```
rd.forward (req, res) throws ServletException, IOException  
rd.include (req, res) throws ServletException, IOException
```

Day -33:

Write a java program which illustrates the concept of servlet chaining?

Answer:

web.xml:

```
<web-app>  
    <servlet>  
        <servlet-name>abc</servlet-name>
```

```
        <servlet-class>Serv1</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>Serv2</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/s1</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>pqr</servlet-name>
        <url-pattern>/s2</url-pattern>
    </servlet-mapping>
</web-app>
```

Serv1.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h2>I AM FROM Serv1 BEGINNING</h2>");
        ServletContext ctx=getServletContext ();
        RequestDispatcher rd=ctx.getRequestDispatcher ("/s2");
        rd.include (req, res);
        pw.println ("<h2>I AM FROM Serv1 ENDING</h2>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

Serv2.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h2>I AM FROM Serv2</h2>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

What is the difference between `getRequestDispatcher (String)` and `getNamedRequestDispatcher (String)`?

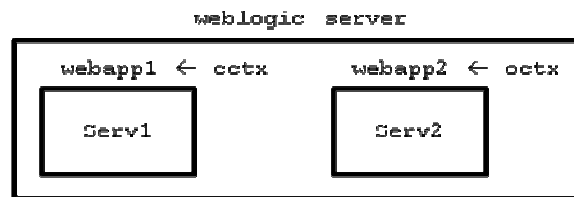
Answer:

`getRequestDispatcher (String)` method takes **url-pattern** or **public-url** of *web.xml* where as `getNamedRequestDispatcher (String)` method takes **name of the servlet** or **deployer name** of *web.xml*

Forwarding or Including request and response of one web-app to another web-app:

In order to achieve forwarding or including the request and response objects of one web application to another web application, we must ensure that both the web applications must run in the same servlet container.

1. Obtain `ServletContext` object for the current web application.



For example:

```
ServletContext cctx=getServletContext ();
```

2. Obtain `ServletContext` object of another web application by using the following method which is present in `ServletContext` interface.

```
javax.servlet.ServletContext  
    ↓  
public ServletContext getContext (String);  
    ↑  
Name of another web application
```

For example:

```
ServletContext octx=cctx.getContext ("/webapp2");
```

3. Obtain `RequestDispatcher` object by using `ServletContext` object of another web application.

```
javax.servlet.ServletContext  
    ↓  
public RequestDispatcher getRequestDispatcher (String);  
    ↑  
url-pattern of another web application
```

For example:

```
RequestDispatcher rd=octx.getRequestDispatcher ("/s2");
```

4. Use either `include` or `forward` to pass request and response objects of current web application.

For example:

```
rd.include (req, res);  
rd.forward (req, res);
```

Day - 34:**Deploying in same servers but from different web applications:****For example:****web.xml** (webapp1):

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>Serv1</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/s1</url-pattern>
    </servlet-mapping>
</web-app>
```

Serv1.java (webapp1):

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h2>I AM FROM Serv1 BEGINNING OF webapp1</h2>");
        ServletContext cctx=getServletContext ();
        ServletContext octx=cctx.getContext ("/webapp2");
        RequestDispatcher rd=octx.getRequestDispatcher ("/s2");
        rd.include (req, res); // rd.forward (req, res);
        pw.println ("<h2>I AM FROM Serv2 ENDING OF webapp1</h2>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

web.xml (webapp2):

```
<web-app>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>Serv2</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>pqr</servlet-name>
        <url-pattern>/s2</url-pattern>
    </servlet-mapping>
</web-app>
```

Serv2.java (webapp2):

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



```

public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h6>I AM FROM Serv2 OF webapp2</h6>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};

```

Forwarding request and response objects of one web application to another web application and both the web applications are running in different servlet containers:

In order to send the request and response object of one web application which is running in on servlet container to another web application which is running in another servlet container, we cannot use forward and include methods.

To achieve the above we must use a method called sendRedirect (String url) method whose prototype is

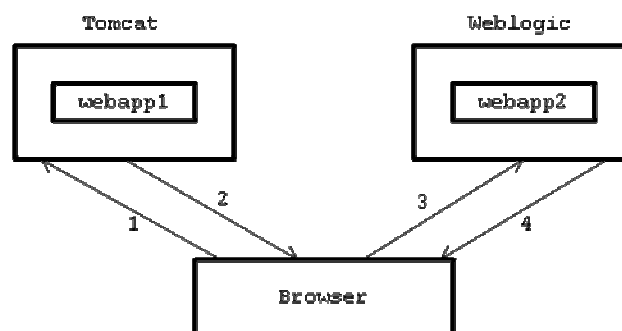
```

javax.servlet.http.HttpServletResponse
    |
    v
    public void sendRedirect (String url);

```

The above method is present in HttpServletResponse interface, the parameter String url represents url of another web application which is running in some other servlet container.

The following diagram will illustrate the concept of sendRedirect method:



1. Make a request to a servlet or JSP which is running in a web application of one container <http://localhost:2008/webapp1/s1> context path or document root of one web application.
2. Servlet of web application of Tomcat will redirect the request of client to another web application of Weblogic by using the following statement:

`Res.sendRedirect ("http://localhost:7001/webapp2/s2");` must be written in Serv1 of webapp1

3. Browser will send the request to another web application of another servlet container.

For example:

<http://localhost:7001/webapp2/s2> which is redirected by Serv1 of webapp1.

4. Webapp1 gives the response of Serv2 only but not Serv1 servlet.

Deploying in different servers:**For example:****web.xml** (webapp1):

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>Serv1</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/s1</url-pattern>
    </servlet-mapping>
</web-app>
```

Serv1.java (webapp1):

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv1 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h2>I AM FROM Serv1 OF webapp1</h2>");
        res.sendRedirect ("http://localhost:7001/webapp2/s2");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

NOTE: webapp1 must be deployed in Tomcat server.

web.xml (webapp2):

```
<web-app>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>Serv2</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>pqr</servlet-name>
        <url-pattern>/s2</url-pattern>
    </servlet-mapping>
</web-app>
```

Serv2.java (webapp2):

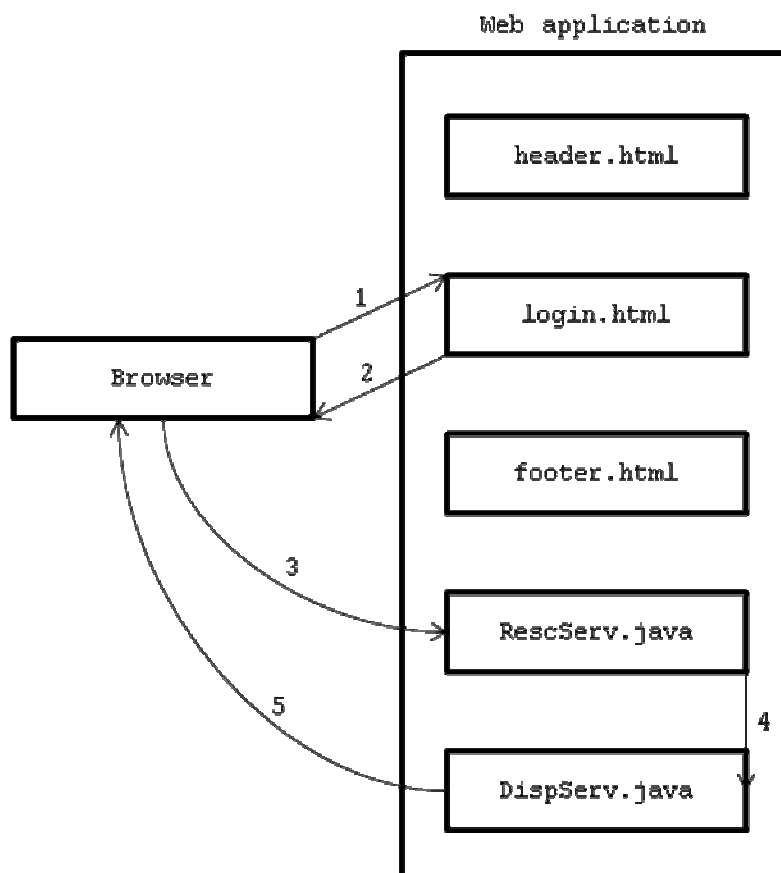
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Serv2 extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
```

```
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        pw.println ("<h6>I AM FROM Serv2 OF webapp2</h6>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

NOTE: webapp2 must be deployed in **Weblogic server**.

Day - 35:

Develop the following application with the help of request dispatcher by using forward and include methods?



Answer:

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>RecvServ</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>DispServ</servlet-class>
    </servlet>
```

header.html:

login.html:

footer.html:

By Mr. K. V. R Page 75

RecvServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class RecvServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        RequestDispatcher rd=req.getRequestDispatcher ("/display");
        rd.forward (req, res);
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

DispServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DispServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String uname=req.getParameter ("header_euname");
        String pwd=req.getParameter ("header_epwd");
        RequestDispatcher rd1=req.getRequestDispatcher ("header.html");
        rd1.include (req, res);
        pw.println ("<br><br><br>");
        if (uname.equals ("kvr") && pwd.equals ("advanced"))
        {
            pw.println ("<center><font color=#ffff66><h3>VALID CREDENTIALS</h3></center>");
        }
        else
        {
            pw.println ("<center><font color=#ffff66><h3>INVALID CREDENTIALS</h3></center>");
        }
        pw.println ("</font></br></br></br>");
        RequestDispatcher rd2=req.getRequestDispatcher ("footer.html");
        rd2.include (req, res);
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

NOTE:

1. Whenever we pass a html program as a parameter to getRequestDispatcher method, that html program will be executed by browser at client side where as whenever we pass a servlet program or a JSP which will be executed by servlet container or server.

2. RequestDispatcher interface object can also be obtained by using the following method which is present in HttpServletRequest.

```
javax.servlet.http.HttpServletRequest  
↓  
public RequestDispatcher getRequestDispatcher (String);
```

Day - 36:

Attribute scopes:

Life time or visibility of a variable or attribute is known as scope. In servlet programming environment we have 3 types of attribute scopes, they are **HttpServletRequest scope**, **ServletContext scope** and **Session scope**.

HttpServletRequest object scope:

The request data is available in an object of HttpServletRequest or ServletRequest in the form of (key, value) form. Since we are using HttpServletRequest object only either in service or doGet or doPost methods, hence we cannot access this object in other methods of same servlet. Therefore the scope of HttpServletRequest object is limited to only service or doGet or doPost methods of a particular servlet.

If the group of Servlets is involving in processing a single client request then all the Servlets can use the HttpServletRequest object in their respective service or doGet or doPost methods.

NOTE:

Whenever we want to send a temporary result or local data to another servlet to be accessed in service or doGet or doPost methods, it is highly recommended to add the temporary data to the request object in the form of (key, value).

Methods:

```
public void setAttribute (String, Object); →1  
public Object getAttribute (String); →2  
public void removeAttribute (String); →3  
public Enumeration getAttributeNames (); →4
```

Method-1 is used for inserting or adding the data to HttpServletRequest object in the form of (key, value) pair. The parameter String represents key and the parameter Object represents value for the key. If the key value is not present in HttpServletRequest object then the data will be added as a new entry. If the key value already present in a HttpServletRequest object then key value remains same and whose value will be modified with the new value.

For example:

```
req.setAttribute ("v1", new Integer (10));  
req.setAttribute ("v2", "abc");
```

Method-2 is used for getting or obtaining the value of value by passing value of key. If the value of key is not found in HttpServletRequest object then its return type (i.e., value of object) is null.

For example:

```
Object obj=req.getAttribute ("uname");
```

Method-3 is used for removing or deleting the (key, value) pair from HttpServletRequest object by passing value of key. If the value of key is not founded in the HttpServletRequest object then nothing is removed from HttpServletRequest object.

For example:

```
req.removeAttribute (uname);
```

Method-4 is used for obtaining the names of keys which are available in HttpServletRequest object.

For example:

```
Enumeration en=req.getAttributeNames ();
```

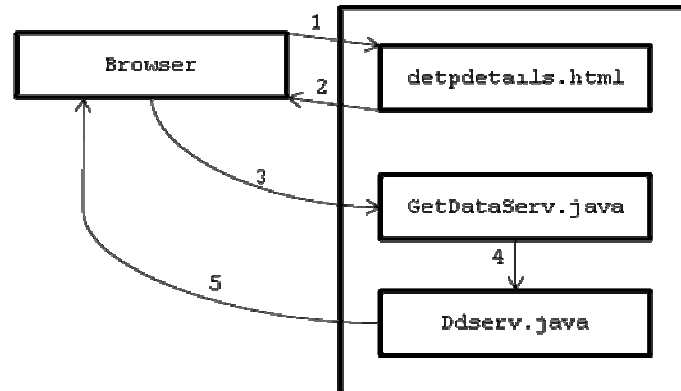


```

While (en.hasMoreElements ())
{
    Object kname=en.nextElement ();
    String key= (String) kname;
    Object val=req.getAttribute (key);
}
  
```

Day -37:

Implement the following diagram:



Answer:

deptdetails.html:

```

<html>
<title>Retrieve department details</title>
<body bgcolor="#EEE8AA">
<center>
<form name="deptdetails" action="./gdserv" method="post">
<table border="1" bgcolor="#FFE4E1">
<tr>
<th>Enter department number : </th>
<td><input type="text" name="deptdetails_no" value=""></td>
</tr>
<tr>
<td align="center"><input type="submit" name="deptdetails_details" value="Details"></td>
<td align="center"><input type="reset" name="deptdetails_reset" value="Reset"></td>
  
```

```
        </tr>
    </table>
</form>
</center>
</body>
</html>
```

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>GetDataServ</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>Ddserv</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/gdserv</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>pqr</servlet-name>
        <url-pattern>/ddserv</url-pattern>
    </servlet-mapping>
</web-app>
```

GetDataServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
public class GetDataServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String dnol=req.getParameter ("deptdetails_no");
        int dno=Integer.parseInt (dnol);
        try
        {
            Class.forName ("oracle.jdbc.driver.OracleDriver");
            Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
Hanuman","scott","tiger");
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from dept where deptno="+dno);
            rs.next ();
            String vdno=rs.getString (1);
            String vname=rs.getString (2);
            String vloc=rs.getString (3);
            req.setAttribute ("sdno",vdno);
            req.setAttribute ("sdname",vname);
            req.setAttribute ("sdloc",vloc);
            ServletContext ctx=getServletContext ();
```



```
        RequestDispatcher rd=ctx.getRequestDispatcher ("/ddserv");
        rd.forward (req, res);
        con.close ();
    }
    catch (Exception e)
    {
        e.printStackTrace ();
    }
}
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    doGet (req, res);
}
};
```

Ddserv.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Ddserv extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String rdno= (String) req.getAttribute ("sdno");
        String rdname= (String) req.getAttribute ("sdname");
        String rdloc= (String) req.getAttribute ("sdloc");
        pw.println ("<h3>"+rdno+"</h3>");
        pw.println ("<h3>"+rdname+"</h3>");
        pw.println ("<h3>"+rdloc+"</h3>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

ServletContext scope:

An object of ServletContext is used for having a common data for a group of servlets which belongs to same web application. The data of ServletContext object can be accessed through out all the methods of the servlet of a particular web.xml

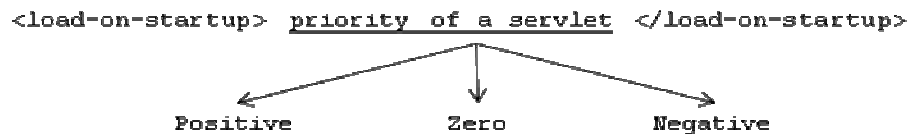
Methods:

The following methods are used to add local data of a servlet to ServletContext object, removing the existing data of ServletContext object.

```
public void setAttribute (String, Object); →1
public Object getAttribute (String); →2
public void removeAttribute (String); →3
public Enumeration getAttributeNames (); →4
```

Load-on-startup:

Load-on-startup is basically used for giving equal response for all the clients who are accessing a particular web application. By default after making request the ServletContext object will be created by servlet container. Because of this first response takes more amount of time and further responses will take minimum amount of time. Therefore to avoid the discrepancy in response time we use a concept of load-on-startup. <load-on-startup> tag will be used as a part of <servlet> tag since it is specific to the servlet.



If the priority value is positive for a group of servlets then whose objects will be created based on ascending order of the priorities. If the priority value is zero then that servlet object will be created at the end. If the priority value of a servlet is negative then that servlet object will not be created i.e., neglected.

When we use load-on-startup as a part of web.xml the container will create an object of a servlet before first request is made.

web.xml:

```
<web-app>
  <servlet>
    <servlet-name>abc</servlet-name>
    <servlet-class>DdServ</servlet-class>
    <load-on-startup>10</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/ddurl</url-pattern>
  </servlet-mapping>
</web-app>
```

DdServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DdServ extends HttpServlet
{
    public DdServ ()
    {
        System.out.println ("SERVLET OBJECT IS CREATED");
    }
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        System.out.println ("I AM FROM doGet ()");
        pw.println ("<h3>I AM FROM doGet ()</h3>");
    }
};
```

Day - 38:**SESSION MANAGEMENT**

Whenever we make a request that request can be processed by group of Servlets and we get single response (in case of servlet chaining).

In general scenario we may make much number of requests to many numbers of Servlets to get many numbers of responses. By default all these 'n' number of requests and responses are independent to each other. Since, the protocol which we are using is *http*. *http* protocol is a **stateless protocol**. A stateless protocol is one which never maintains an identity of the client forever.

In order to make 'n' number of independent request and responses as a consecutive request and responses we must use the concept of **session management** or **session tracking**.

Session management is a process of maintaining an identity of the client for a period of time for multiple requests to get multiple responses across the network.

Session management techniques:

In order to maintain an identity of the client for a period of time, we have four types of session management techniques; they are **Cookies**, **HttpSession**, **Hidden form field** and **URL rewritten**.

COOKIES

A *cookie* is the piece of information which contains an identity of the client in the form of (key, value) pair. Key always represents cookie name and value represents value of the cookie.

A Cookie is the class developed according to http protocol specification for maintaining for identity of client and it is present in a package called `javax.servlet.http.*` package.

Steps for developing Cookie:

1. Create an object of a Cookie.

```
Cookie ck=new Cookie (String, Object);
```

For example:

```
Cookie c1=new Cookie ("fno","10");  
Cookie c2=new Cookie ("sno","20");
```

2. Every cookie must be added as a part of response (add the cookie to response object).

For example:

```
res.addCookie (c1);  
res.addCookie (c2);
```

Here, c1 and c2 are Cookie class objects created in step-1 and addCookie () is an instance method present in HttpServletResponse interface.

```
javax.servlet.http.HttpServletResponse  
↓  
public void addCookie (Cookie);
```

3. In order to get the cookies we must use the following method which is present in HttpServletRequest.

```
javax.servlet.http.HttpServletRequest  
└──  
    public Cookie [] get_cookies ();
```

For example:

```
Cookie ck []=req.getCookies ();  
if (ck!=null)  
{  
    pw.println ("COOKIES ARE PRESENT");  
}  
else  
{  
    pw.println ("COOKIES ARE NOT PRESENT");  
}
```

Day - 39:

4. In order to obtain cookie name, cookie value and to set its age we have to use the following methods:

```
public String getName ();  
public Object getValue ();  
public void setMaxAge (long sec);  
public long getMaxAge ();
```

Methods 1 and 2 are used for obtaining name and value of the cookie. Methods 3 and 4 are used for setting and obtaining the age of the cookie.

The **default age** of the **cookie** will be **-1** and it will be available only for current browsing session and a cookie will not be available when the browser is closed, the system is rebooted. **Cookies are prepared by server side program and there will be residing in client side.**

For example:

```
Cookie c1=new Cookie ();  
c1.setMaxAge (24*60*60); // setting the cookie age for 24 hours.  
String s=c1.getName ();  
Object obj=c1.getValue ();
```

Write a java program which illustrates the concept of setting and getting cookies by specifying maximum age, default age and obtaining the cookies which are present at client side?

Answer:**web.xml:**

```
<web-app>  
    <servlet>  
        <servlet-name>abc</servlet-name>  
        <servlet-class>SetCookie</servlet-class>  
    </servlet>  
    <servlet>  
        <servlet-name>pqr</servlet-name>  
        <servlet-class>ShowCookie</servlet-class>  
    </servlet>
```

```
<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/test1</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>pqr</servlet-name>
    <url-pattern>/test2</url-pattern>
</servlet-mapping>
</web-app>
```

SetCookie.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SetCookie extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        // default maximum age is -1, indicating cookie applies only to current browsing session
        res.setContentType ("text/html");
        Cookie c1=new Cookie ("ANDHRA PRADESH", "HYDERABAD");
        Cookie c2=new Cookie ("TAMILNADU", "CHENNAI");
        res.addCookie (c1);
        res.addCookie (c2);
        // c3 is valid for 5mins & c4 for 10mins, regardless of user quits browser, reboots computer
        Cookie c3=new Cookie ("KARNATAKA", "BANGLORE");
        Cookie c4=new Cookie ("BIHAR", "PATNA");
        c3.setMaxAge (300);
        c4.setMaxAge (600);
        res.addCookie (c3);
        res.addCookie (c4);
        System.out.println ("SUCCESSFUL IN SETTING COOKIES");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

ShowCookie.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ShowCookie extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String title="Active Cookies";
        pw.println ("<html><head><title>"+title+"</title></head></body>");
        pw.println ("<table border=\"1\" align=\"center\">");
        pw.println ("<tr><th>Cookie Name</th><th>Cookie Value</th></tr>");
        Cookie ck []=req.getCookies ();
        if (ck!=null)
```

```

    {
        for (int i=0; i<ck.length; i++)
        {
            pw.println("<tr><td>"+ck[i].getName()+"</td><td>"+ck[i].getValue()+"</td></tr>");
        }
    }
    else
    {
        System.out.println ("NO COOKIES PRESENT");
    }
    pw.println ("</table></body></html>");
}
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    doGet (req, res);
}
};

```

Day - 40:

Implement the following screen by using cookies?

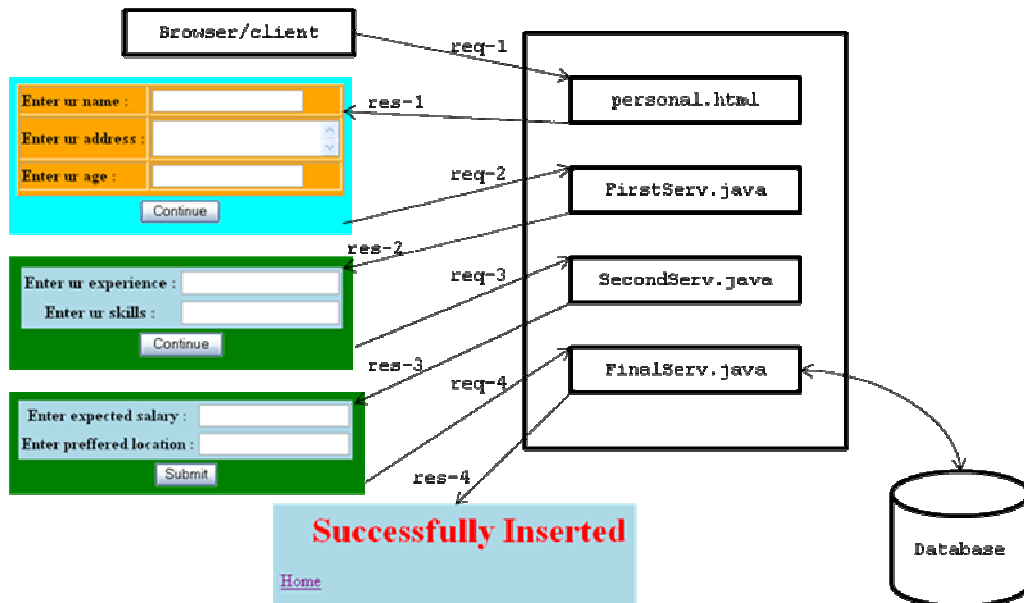


Figure -1

Answer:**web.xml:**

```

<web-app>
    <servlet>
        <servlet-name>s1</servlet-name>
        <servlet-class>FirstServ</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>s2</servlet-name>
        <servlet-class>SecondServ</servlet-class>
    </servlet>
    <servlet>

```

```
        <servlet-name>s3</servlet-name>
        <servlet-class>FinalServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>s1</servlet-name>
        <url-pattern>/test1</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>s2</servlet-name>
        <url-pattern>/test2</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>s3</servlet-name>
        <url-pattern>/test3</url-pattern>
    </servlet-mapping>
</web-app>
```

personal.html:

```
<html>
  <title>Complete example</title>
  <body>
    <center>
      <form name="ex" action="./test1" method="post">
        <table border="1">
          <tr>
            <th align="left">Enter ur name : </th>
            <td><input type="text" name="ex_name" value=""></td>
          </tr>
          <tr>
            <th align="left">Enter ur address : </th>
            <td><textarea name="ex_add" value=""></textarea></td>
          </tr>
          <tr>
            <th align="left">Enter ur age : </th>
            <td><input type="text" name="ex_age" value=""></td>
          </tr>
          <tr>
            <td>
              <table>
                <tr>
                  <td align="center"><input type="submit" name="ex_continue" value="Continue"></td>
                </tr>
              </table>
            </td>
          </tr>
        </table>
      </form>
    </center>
  </body>
</html>
```

FirstServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class FirstServ extends HttpServlet
{
```

```

public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    res.setContentType ("text/html");
    PrintWriter pw=res.getWriter ();
    String name=req.getParameter ("ex_name");
    String address=req.getParameter ("ex_add");
    String age=req.getParameter ("ex_age");
    Cookie c1=new Cookie ("name",name);
    Cookie c2=new Cookie ("address",address);
    Cookie c3=new Cookie ("age",age);
    res.addCookie (c1);
    res.addCookie (c2);
    res.addCookie (c3);
    pw.println ("<html><title>First Servlet</title><body bgcolor=\"green\"><center>");
    pw.println ("<form name=\"first\" action=\"./test2\" method=\"post\"><table bgcolor=\"lightblue\">");
    pw.println ("<tr><th>Enter ur experience : </th><td><input type=\"text\" name=\"first_exp\"
value=\"\">");
    pw.println ("</td></tr><tr><th>Enter ur skills : </th><td><input type=\"text\" name=\"first_skills\"
value=\"\">");
    pw.println ("</td></tr><table><tr><td align=\"center\"><input type=\"submit\" name=\"first_submit\"
value=\"Continue\">");
    pw.println ("</td></tr></table></table></form></center></body></html>");
}
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    doGet (req, res);
}
};

```

SecondServ.java:

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SecondServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String exp=req.getParameter ("first_exp");
        String skills=req.getParameter ("first_skills");
        Cookie c4=new Cookie ("exp",exp);
        Cookie c5=new Cookie ("skills",skills);
        res.addCookie (c4);
        res.addCookie (c5);
        pw.println ("<html><title>Second Servlet</title><body bgcolor=\"green\"><center>");
        pw.println ("<form name=\"second\" action=\"./test3\" method=\"post\"><table
bgcolor=\"lightblue\">");
        pw.println ("<tr><th>Enter expected salary : </th><td><input type=\"text\" name=\"second_sal\"
value=\"\">");
        pw.println ("</td></tr><tr><th>Enter preffered location : </th><td><input type=\"text\"
name=\"second_loc\" value=\"\">");
        pw.println ("</td></tr><table><tr><td align=\"center\"><input type=\"submit\"
name=\"second_submit\" value=\"Submit\">");
        pw.println ("</td></tr></table></table></form></center></body></html>");
    }
}

```



```
    }  
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
    {  
        doGet (req, res);  
    }  
};
```

Database table (info):

```
create table info  
(  
    name varchar2 (13),  
    addr varchar2 (33),  
    age number (2),  
    exp number (2),  
    skills varchar2 (13),  
    sal number (7,2),  
    loc varchar2 (17)  
);  
/
```

FinalServ.java:

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import java.sql.*;  
import java.util.*;  
public class FinalServ extends HttpServlet  
{  
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
    {  
        ArrayList al=new ArrayList ();  
        res.setContentType ("text/html");  
        PrintWriter pw=res.getWriter ();  
        String sal=req.getParameter ("second_sal");  
        float salary=Float.parseFloat (sal);  
        String location=req.getParameter ("second_loc");  
        Cookie ck []=req.getCookies ();  
        for (int i=0; i<ck.length; i++)  
        {  
            al.add (ck[i].getValue ());  
        }  
        String name=al.get (0).toString ();  
        String address=al.get (1).toString ();  
        String age1=al.get (2).toString ();  
        int age=Integer.parseInt (age1);  
        String exp=al.get (3).toString ();  
        int experience=Integer.parseInt (exp);  
        String skills=al.get (4).toString ();  
        try  
        {  
            Class.forName ("oracle.jdbc.driver.OracleDriver");  
            Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:  
Hanuman","scott","tiger");  
            PreparedStatement ps=con.prepareStatement ("insert into info values (?, ?, ?, ?, ?)");  
            ps.setString (1, name);
```

```
        ps.setString (2, address);
        ps.setInt (3, age);
        ps.setInt (4, experience);
        ps.setString (5, skills);
        ps.setFloat (6, salary);
        ps.setString (7, location);
        int j=ps.executeUpdate ();
        if (j>0)
        {
            pw.println ("<html><body bgcolor=\"lightblue\"><center><h1><font color=\"red\">
Successfully ");
            pw.println ("Inserted</font></h1></center><a href=\"personal.html\">Home</a>
</body></html>");
        }
        else
        {
            pw.println ("<html><body bgcolor=\"cyan\"><center><h1><font color=\"red\">Try
Again");
            pw.println ("</font></h1></center><a href=\"personal.html\">Home</a></body>
</html>");
        }
    }
    catch (Exception e)
    {
        pw.println ("<html><body bgcolor=\"cyan\"><center><h1><font color=\"red\">Try Again");
        pw.println ("</font></h1></center><a href=\"personal.html\">Home</a></body></html>");
    }
}
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    doGet (req, res);
}
};
```

Day - 41:

Disadvantages of Cookies:

1. When we remove the cookies which are residing in client side by going to `tools` → `internet options` → `delete cookies` of browser, we cannot maintain identity of the client.
2. There is a restriction on size of the cookies (i.e., `20 cookies` are permitted per web application).
3. As and when number of cookies which leads to more network traffic flow and there is a possibility of loosing performance of server side applications.

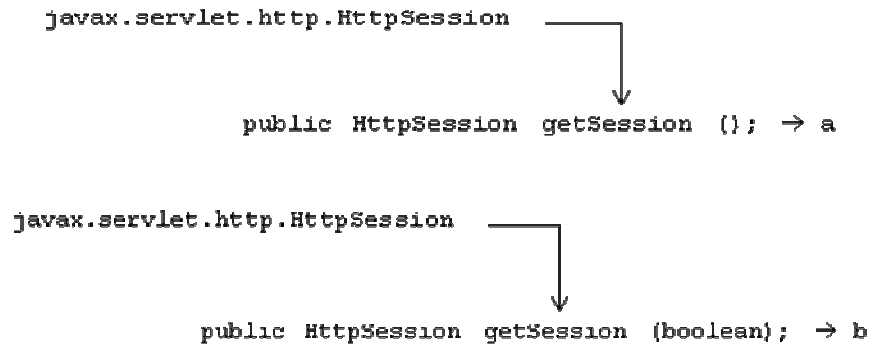
HTTP SESSION

HttpSession is a technique of session management which maintains an identity of a client for a period of time across the network for making 'n' number of requests for obtaining 'n' number of responses. During the period of session, all the requests and responses are consecutive.

Day - 42:**Steps for developing *HttpSession* applications:**

1. Obtain an object of `javax.servlet.http.HttpSession` interface.

In order to obtain an object of *HttpSession* we have to use the following two methods which are present in `HttpServletRequest`.

**For example:**

```
HttpSession hs=req.getSession ();
HttpSession hs=req.getSession (true);
```

When we use method-ii the boolean value can be either true or false:

Case-1: Session **not created** and boolean value is **true**. Session will be created newly.

For example:

```
HttpSession hs=req.getSession (true);
```

Case-2: Session **not created** and boolean value is **false**. Session will not be created at all.

For example:

```
HttpSession hs=req.getSession (false);
```

Case-3: Session **already created** and boolean value is **true**. Existing or old session will be continued with checking.

For example:

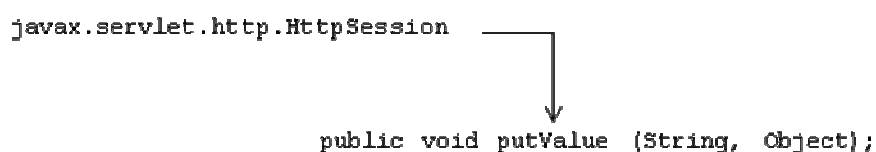
```
HttpSession hs=req.getSession (true);
```

Case-4: Session **already created** and boolean value is **false**. Existing or old session will be continued without any checking.

For example:

```
HttpSession hs=req.getSession (false);
```

2. Put the values into *HttpSession* object by using the following methods which are present in *HttpSession*:



```
javax.servlet.http.HttpSession  
↓  
public void setAttribute (String, Object);
```

Here, String represents session variable name known as key and Object represents session value. In *HttpSession* object the data is organizing in the form of (key, value) pair.

For example:

```
hs.putValue ("name", "kalyan");  
hs.setAttribute ("address", s2);
```

3. Get the values from *HttpSession* object by using the following methods which are present in *HttpSession*:

```
javax.servlet.http.HttpSession  
↓  
public Object getValue (String);
```

```
javax.servlet.http.HttpSession  
↓  
public Object getAttribute (String);
```

For example:

```
Object obj1=hs.getAttribute ("name");  
Object obj2=hs.getValue ("address");
```

Other methods in HttpSession:

1. public void setMaxInactiveInterval (long sec):

In most of the popular web sites an identity of the client will be maintained for a period of 30 minutes. When the time interval between first request and second request i.e., if the time delay between one request to another request is 30 minutes then automatically the server will eliminate the identity of the client.

In order to set our own session out time or maximum session active time, we use this method. In this method, we specify the session active time in terms of seconds.

For example:

```
hs.setMaxInactiveInterval (60*60);
```

2. public boolean isNew ():

This method returns true when the session is created newly otherwise it returns false for old sessions.

For example:

```
boolean b=hs.isNew ();
```

3. public void removeAttribute (String):

This method is used for removing one of the session attribute name along with its value.

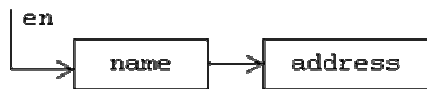
For example:

```
hs.removeAttribute ("address");
```

4. **public Enumeration *getAttributeNames* ()**: This method is used for obtaining all the session variable names.

For example:

```
Enumeration en=hs.getAttributeNames ();
```



```
While (en.hasMoreElements ())
{
    Object kname=en.nextElement ();
    String key= (String) kname;
    Object val=req.getAttribute (key);
}
```

5. **public void invalidate ()**:

This method is used for invalidate the identity of the client permanently i.e., all the values of session object will be removed completely.

For example:

```
hs.invalidate ();
```

6. **public long *getLastAccessTime* ()**: This method is used for obtaining last access time.

For example:

```
long t=hs.getLastAccessTime ();
```

7. **public long getId ()**:

This method is used for obtaining session id of a client which is created by servlet container.

For example:

```
long sid=hs.getId ();
```

Day - 43:

Develop the servlets which illustrate the concept of HttpSession? [Refer above figure-1]

Answer:

web.xml:

```
<web-app>
    <servlet>
        <servlet-name>s1</servlet-name>
        <servlet-class>FirstServ</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>s2</servlet-name>
        <servlet-class>SecondServ</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>s3</servlet-name>
        <servlet-class>FinalServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>s1</servlet-name>
        <url-pattern>/test1</url-pattern>
```

```
</servlet-mapping>
<servlet-mapping>
    <servlet-name>s2</servlet-name>
    <url-pattern>/test2</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>s3</servlet-name>
    <url-pattern>/test3</url-pattern>
</servlet-mapping>
</web-app>
```

personal.html:

```
<html>
<title>Complete example</title>
<body>
    <center>
        <form name="ex" action="./test1" method="post">
            <table border="1">
                <tr>
                    <th align="left">Enter ur name : </th>
                    <td><input type="text" name="ex_name" value=""></td>
                </tr>
                <tr>
                    <th align="left">Enter ur address : </th>
                    <td><textarea name="ex_add" value=""></textarea></td>
                </tr>
                <tr>
                    <th align="left">Enter ur age : </th>
                    <td><input type="text" name="ex_age" value=""></td>
                </tr>
                <tr>
                    <td colspan="2">
                        <table>
                            <tr>
                                <td align="center"><input type="submit" name="ex_continue" value="Continue"></td>
                            </tr>
                        </table>
                    </td>
                </tr>
            </table>
        </form>
    </center>
</body>
</html>
```

Database table (info):

```
create table info
(
    name varchar2 (13),
    addr varchar2 (33),
    age number (2),
    exp number (2),
    skills varchar2 (13),
    sal number (7,2),
    loc varchar2 (17)
);
/
```

FirstServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class FirstServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String name1=req.getParameter ("ex_name");
        String address1=req.getParameter ("ex_add");
        String age1=req.getParameter ("ex_age");
        HttpSession hs=req.getSession (true);
        hs.putValue ("namehs",name1); // we can also use setAttribute
        hs.putValue ("addresshs",address1); // we can also use setAttribute
        hs.putValue ("agehs",age1);
        pw.println ("<html><title>First Servlet</title><body bgcolor=\"green\"><center>");
        pw.println ("<form name=\"first\" action=\"./test2\" method=\"post\"><table bgcolor=\"lightblue\">");
        pw.println ("<tr><th>Enter ur experience : </th><td><input type=\"text\" name=\"first_exp\"
value=\"\">");
        pw.println ("</td></tr><tr><th>Enter ur skills : </th><td><input type=\"text\" name=\"first_skills\"
value=\"\">");
        pw.println ("</td></tr><table><tr><td align=\"center\"><input type=\"submit\" name=\"first_submit\"
value=\"Continue\">");
        pw.println ("</td></tr></table></table></form></center></body></html>");
    }
    public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        doGet (req, res);
    }
};
```

SecondServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SecondServ extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        String exp1=req.getParameter ("first_exp");
        String skills1=req.getParameter ("first_skills");
        HttpSession hs=req.getSession (false);
        hs.putValue ("exphs",exp1); // we can also use setAttribute
        hs.putValue ("skillshs",skills1); // we can also use setAttribute
        pw.println ("<html><title>Second Servlet</title><body bgcolor=\"green\"><center>");
        pw.println ("<form name=\"second\" action=\"./test3\" method=\"post\"><table
bgcolor=\"lightblue\">");
        pw.println ("<tr><th>Enter expected salary : </th><td><input type=\"text\" name=\"second_sal\"
value=\"\">");
        pw.println ("</td></tr><tr><th>Enter preffered location : </th><td><input type=\"text\"
name=\"second_loc\" value=\"\">");
    }
```

```
                pw.println ("</td></tr><table><tr><td align=\"center\"><input type=\"submit\"  
name=\"second_submit\" value=\"Submit\">");  
                pw.println ("</td></tr></table></table></form></center></body></html>");  
            }  
            public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
            {  
                doGet (req, res);  
            }  
};
```

FinalServ.java:

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import java.sql.*;  
import java.util.*;  
public class FinalServ extends HttpServlet  
{  
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
    {  
        res.setContentType ("text/html");  
        PrintWriter pw=res.getWriter ();  
        String sal=req.getParameter ("second_sal");  
        float salary=Float.parseFloat (sal);  
        String location=req.getParameter ("second_loc");  
  
        HttpSession hs=req.getSession (false);  
  
        String name=(String) hs.getAttribute ("namehs");  
        String address=(String) hs.getAttribute ("addresshs");  
        String age1=(String) hs.getAttribute ("agehs");  
        int age=Integer.parseInt (age1);  
        String exp=(String) hs.getAttribute ("exphs");  
        int experiance=Integer.parseInt (exp);  
        String skills=(String) hs.getAttribute ("skillshs");  
        try  
        {  
            Class.forName ("oracle.jdbc.driver.OracleDriver");  
            Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:  
Hanuman","scott","tiger");  
            PreparedStatement ps=con.prepareStatement ("insert into info values (?, ?, ?, ?, ?, ?)");  
            ps.setString (1, name);  
            ps.setString (2, address);  
            ps.setInt (3, age);  
            ps.setInt (4, experiance);  
            ps.setString (5, skills);  
            ps.setFloat (6, salary);  
            ps.setString (7, location);  
            int j=ps.executeUpdate ();  
            if (j>0)  
            {  
                pw.println ("<html><body bgcolor=\"lightblue\"><center><h1><font color=\"red\">  
Successfully");  
                pw.println ("Inserted</font></h1></center><a href=\"personal.html\">Home</a>  
</body></html>");  
            }  
        }  
    }  
}
```



```
        }
        else
        {
            pw.println ("<html><body bgcolor=\"cyan\"><center><h1><font color=\"red\">Try
Again");
            pw.println ("</font></h1></center><a href=\"personal.html\">Home</a></body>
</html>");
        }
    }
    catch (Exception e)
    {
        pw.println ("<html><body bgcolor=\"cyan\"><center><h1><font color=\"red\">Try Again");
        pw.println ("</font></h1></center><a href=\"personal.html\">Home</a></body></html>");
    }
}
public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
    doGet (req, res);
}
};
```

Advantages of HttpSession:

1. There is **no restriction on size** of HttpSession object. Since, a single object can hold different type of values which are related to client identity.
2. **Network traffic** flow is very less. Since, only session id is exchanging between client and server.
3. An object of HttpSession neither exists at client side not exists at server.

HIDDEN FLOW FIELDS

Hidden form fields are also a kind of session management technique which allows us to maintain an identity of the client for a period of time.

```
<input type="hidden" name="name of the hidden component" value="value of the hidden component">
```

For example:

```
<input type="hidden" name="age" value="req.getParameter ("age1") ">
<input type="hidden" name="skills" value="J2EE">
```

Day - 44:

JSP (JAVA SERVER PAGES)

JSP is the technology and whose specification is implemented by server vendors. JSP is an alternative technology for servlets. Since, servlets having the following limitations:

Phases in JSP:

Whenever we write in a JSP page, that JSP page will undergo the following phases:

1. **Translation phase:** It is one which converts **.jsp** program into **.java** program internally by the **container**. Once the translation phase is completed the entire JSP program is available into a pure java program.

2. **Compilation phase:** It is one which converts **.java** program into **.class** file provided no errors found in **.java** program by the container. If errors are found by the container in **.java** program those errors will be listed in the browser.
3. **Execution or Running phase:** It is the process of executing **.class** file by the container.

Limitations of servlets

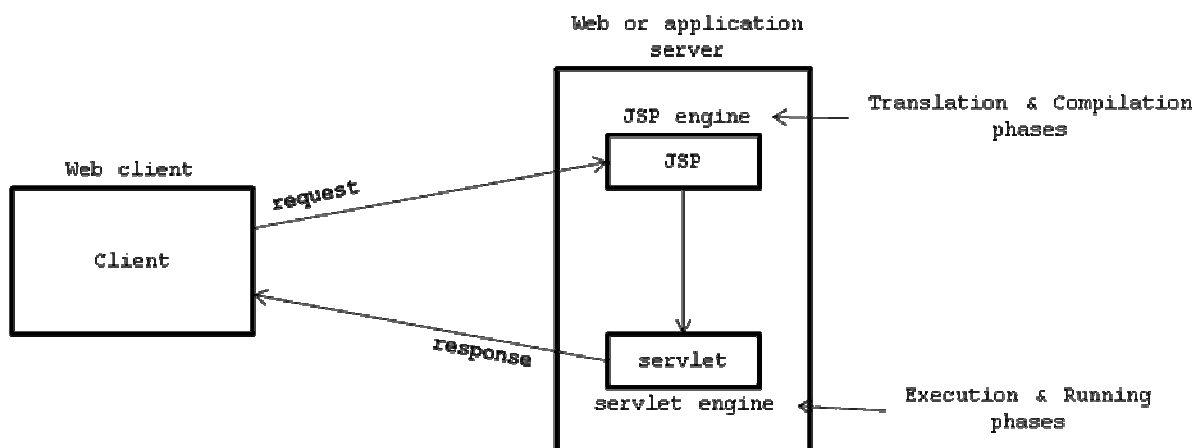
1. In order to develop any servlet they must know java language.
2. Servlets provides uncomfoting. Since, in a single servlet we are writing presentation logic and application or business logic.
3. Maintenance and deployment problems are more (servlets allows only static changes).
4. **Important:** Servlets does not provide automatic page compilation i.e., every servlet program must be compiled explicitly.
5. Servlets does not provide any **custom tag** (user defined tag) development.
6. Servlets does not provide any implicit objects.

JSP provides

1. To write any JSP program java language is not necessary i.e., a non-java programmer can write the JSP page effectively.
2. In JSP there is a separation between presentation logic and application or business logic.
3. JSP page can minimize or reduce maintenance and deployment problems. Since, it allows dynamic changes.
4. JSP provides automatic page compilation (every JSP program internally **translated** as a java program by the container which is **nothing but servlet**).
5. JSP provides the features to develop custom tags.
6. Every JSP page contains 'n' number of implicit objects such as **out, session, exception**, etc.

When we make very first request to a JSP page, **translation** phase, **compilation** phase and **execution** phase will be taken place. From second request to further sub sequent requests only execution phase taking place provided when we are not modifying JSP page.

JSP architecture:



Day - 45:**Life cycle methods of JSP:**

Since, JSP is the server side dynamic technology and it extends the functionality of web or application server, it contains the following life cycle methods:

```
public void jspInit ();  
public void jspService (ServletRequest, ServletResponse);  
public void jspDestroy ();
```

The above three life cycle methods are exactly similar to life cycle methods of servlet.

TAGS in JSP

Writing a program in JSP is nothing but making use of various tags which are available in JSP. In JSP we have three categories of tags; they are **scripting elements, directives and standard actions**.

SCRIPTING ELEMENTS:

Scripting elements are basically used to develop preliminary programming in JSP such as, declaration of variables, expressions and writing the java code. Scripting elements are divided into three types; they are **declaration tag, expression tag and scriptlet**.

1. Declaration tag:

Whenever we use any variables as a part of JSP we have to use those variables in the form of declaration tag i.e., declaration tag is used for declaring the variables in JSP page.

Syntax:

```
<%! Variable declaration or method definition %>
```

When we declare any variable as a part of declaration tag those variables will be available as data members in the servlet and they can be accessed through out the entire servlet.

When we use any methods definition as a part of declaration tag they will be available as member methods in servlet and it will be called automatically by the servlet container as a part of service method.

For example-1:

```
<%!    int a=10, b=30, c;           %>
```

For example-2:

```
<%!  
    int count ()  
    {  
        return (a+b);  
    }  
%>
```

2. Expression tag:

Expression tags are used for writing the java valid expressions as a part of JSP page.

Syntax:

```
<%= java valid expression %>
```

Whatever the expression we write as a part of expression tags that will be given as a response to client by the servlet container. All the expression we write in expression tag they

will be placed automatically in `out.println ()` method and this method is available as a part of service method.

NOTE: Expressions in the expression tag should not be terminated by semi-colon (;).

For example-1:

```
<%! int a=10, b=20 %>
<%= a+b %>
```

The equivalent servlet code for the above expression tag is `out.println (a+b);` out is implicit object of *JSPWriter* class.

For example-2:

```
<%= new java.util.Date () %> ➔ out.println (new java.util.Date ());
```

3. Scriptlet tag:

Scriptlets are basically used to write a pure java code. Whatever the java code we write as a part of scriptlet, that code will be available as a part of `service ()` method of servlet.

Syntax:

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

Write a scriptlet for generating current system date?

Answer:

web.xml:

```
<web-apps>
</web-apps>
```

DateTime.java:

```
<html>
  <title>Current Date & Time</title>
  <head><h4>Current date & time</h4></head>
  <body>
    <%
      Date d=new Date ();
      String s=d.toString ();
      out.println (s);
    %>
  </body>
</html>
```

[or]

```
<html>
  <title>Current Date & Time</title>
  <head><h4>Current date & time</h4></head>
  <body>
    <%=new Date ()%>
  </body>
</html>
```

Write a JSP page to print 1 to 10 numbers? [For *web.xml* refer page no: 102]

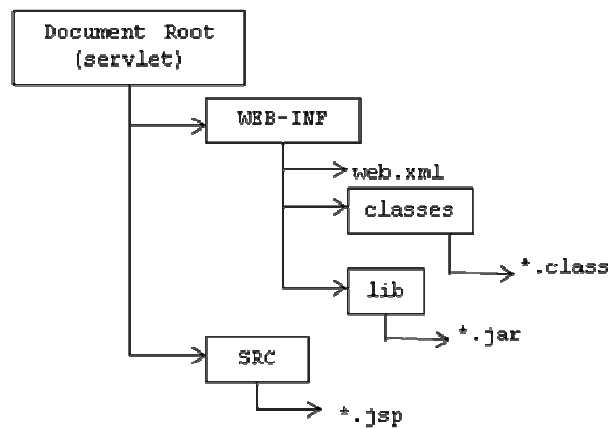
Answer:

One2TenNumbers.jsp:

```
<html>
  <title>Print Numbers 1-10</title>
  <head>Numbers 1-10</head><br>
```

```
<body>
  <%
    for (int i=1; i<=10; i++)
    {
      out.println (i);
    }
  %>
</body>
</html>
```

In order to execute any JSP program one must follow the following directory structure:



Write JSP program to print “Hello JSP world to KALYAN”? [For *web.xml* refer page no: 102]

Answer:

HelloJSP.jsp:

```
<html>
  <title>Fisrt Trail</title>
  <head>Fresher to JSP</head><br>
  <body>
    <h3>Hello JSP world to KALYAN</h3>
  </body>
</html>
```

NOTE:

Whenever we deploy a JSP application in **webapps** folder of **Tomcat** we get an appropriate equivalent servlet for the corresponding JSP file. For example, when we deploy **first.jsp** through a *document root* **first** in **webapps** folder of tomcat we get **first_jsp.java** (which is nothing but a servlet) and **first_jsp.class** by Tomcat server.

The location of servlet and .class file is as follows:

Write a JSP page which will display current data and time? [For *web.xml* refer page no: 102]

Answer:

DateTime.jsp:

```
<html>
  <title>Current Date & Time</title>
  <head>Date & Time without using out.println</head><br>
  <body>
    <%
```

```
        java.util.Date d=new java.util.Date ();
    %>
    <h4>Current date & time</h4>
    <h3><%= d %></h3>
</body>
</html>
```

Write a JSP page which will display number of times a request is made [write a JSP for hit counter]?
[For *web.xml* refer page no: 102]

Answer:

ReloadPageCount.jsp:

```
<html>
    <title>Number of Reloads</title>
    <head>Number of visitings to a browser</head><br>
    <body>
        <%! int ctr=0; %>
        <%!
            int count ()
            {
                return (++ctr);
            }
        %>
        <h3><%= count () %></h3>
    </body>
</html>
```

NOTE:

Within servlet we use to write html code to generate presentation logic whereas in JSP environment within html program we are making use of JSP tags.

Write a JSP page which will retrieve the data from database? [For *web.xml* refer page no: 102]

Answer:

```
<%@ page import="java.sql.*, java.io.*" %>
<html>
    <title>Data From Database</title>
    <head>Retrieve data from Datebase</head>
    <body>
        <%!
            Connection con=null;
            Statement st=null;
            public void jspInit ()
            {
                try
                {
                    Class.forName ("oracle.jdbc.driver.OracleDriver");
                    con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:Hanuman","scott",
"tiger");

                    st=con.createStatement ();
                }
                catch (Exception e)
                {
                    out.println (e);
                }
            }
        %>
    </body>
</html>
```

```
%>
<%!
    try
    {
        ResultSet rs=st.executeQuery ("select * from employee");
        while (rs.next ())
        {
            out.println ("<h3>" + rs.getString (1) + "      " + rs.getString (2) + "</h3>")
        }
    }
    catch (Exception e)
    {
        out.println (e);
    }
%>
</body>
</html>
```

IMPLICIT OBJECTS

Implicit objects are those which will be available to each and every JSP page by default. In JSP we have the following implicit objects.

<u>Implicit object</u>	<u>Creator or Instantiated by</u>
1. out	1. JSPWriter extends <i>PrintWriter</i>
2. request	2. HttpServletRequest
3. response	3. HttpServletResponse
4. application	4. ServletContext
5. config	5. ServletConfig
6. pageContext	6. PageContext
7. page	7. Object (core java example is <i>this</i>)
8. session	8. HttpSession
9. exception	9. Throwable

Day - 46:

DIRECTIVES:

Directives are basically used **to configure the code** that is generated by container in a servlet. As a part of JSP we have three types of directives; they are **page directives**, **include directives** and **taglib directives**.

1. Page directives:

Page directives are basically used for supplying compile time information to the container for generating a servlet. The page directive will take the data in the form of (key, value) pair.

```
<%@
    page  attribute_name1=attribute_value1,
         attribute_name2=attribute_value2,
         ..... ,
         Attribute_nameN=attribute_valueN
%>
```

- Whenever we use page directive as a part of JSP program that statement must be the first statement.
- The **scope of page directive** is applicable to **current JSP page** only.

The following table gives the page directive attribute name and page directive attribute value:

Attribute name	Attribute value
1. import	1. This attribute is used for importing either pre-defined or user-defined packages. The default value is <i>java.lang.*</i> . For example: <code><%@ page import="java.sql.*, java.io.*" %></code>
2. contentType	2. This attribute is used for setting the MIME type (plain/text, img/jpeg, img/gif, audio/wave, etc.). The default value is <i>text/html</i> . For example: <code><%@ page contentType="img/jpeg" %></code>
3. language	3.6 This attribute represents by default <i>java</i> i.e., in order to represent any business logic a JSP program is making use of java language. The attribute language can support any of the other programming languages for developing a business logic at server side. For example: <code><%@ page language="java" %></code>
4.6 isThreadSafe	4. This attribute represents by default <i>true</i> which represents one server side resource can be accessed by many number of clients (each client is treated as one thread). At a time if we make the server side resource to be accessed by a single client then the value of isThreadSafe is <i>false</i> . For example: <code><%@ page isThreadSafe="false" %></code>
5. isErrorPage 6. errorPage	5.6 When we write 'n' number of JSP pages, there is a possibility of occurring exceptions in each and every JSP page. It is not recommended for the JSP programmer to write try and catch blocks in each and every JSP page. It is always recommended to handle all the exceptions in a single JSP page. <i>isErrorPage</i> is an attribute whose default value is <i>true</i> which indicates exceptions to be processed in the same JSP page which is not recommended. If <i>isErrorPage</i> is <i>false</i> then exceptions are not processed as a part of current JSP page and the exceptions are processed in some other JSP page which will be specified through an attribute called <i>errorPage</i> . For example: <code><%@ page isErrorPage="false" errorPage="err.jsp" %></code> <u>err.jsp:</u> <code><%= exception %></code> [or] <code><%= exception.getMessage () %></code>

Day - 47:

7. autoflush 8. buffer	7. Whenever the server side program want to send large amount of data to a client, it is recommended to make <i>autoflush</i> value as <i>false</i> and we must specify <i>the size of the buffer in terms of kb</i> . The default value of <i>autoflush</i> is <i>true</i> which represents the server side program gives the response back to the client each
---------------------------	--

	<p>and every time. Since, the <i>buffer</i> size is zero.</p> <p>For example:</p> <pre><%@ page autoflush="false" buffer="12kb" %> <%@ page autoflush="true" %> [by default]</pre>
9. session	<p>9. When we want to make 'n' number of independent requests as consecutive requests one must use the concept of <i>session</i>. In order to maintain the session we must give the value of <i>session</i> attribute has <i>true</i> in each and every JSP page (recommended).</p> <p>The default value of <i>session</i> is <i>true</i> which represents the <i>session</i> is applicable to current JSP page.</p> <p>For example:</p> <pre><%@ page session="true" %> → session will be created or old session will be continued.</pre>
10. info	<p>10. Using this attribute it is recommended for the JSP programmer to specify <i>functionality about a JSP page</i>, on what <i>date it is created</i> and <i>author</i>.</p> <p>In order to get information (<i>info</i>) of a JSP page we must use the following method:</p> <pre>javax.servlet.Servlet ↓ public String getServletInfo ();</pre>

Write a JSP page which illustrates the concept of `isErrorPage` and `errorPage`?

Answer:

web.xml:

```
<web-app>
</web-app>
```

Exception.jsp:

```
<%@ page isErrorPage="false" errorPage="ErrorPage.jsp" %>
<html>
    <body>
        <%= 30/0 %>
    </body>
</html>
```

ErrorPage.jsp:

```
<%@ page isErrorPage="true" %>
<html>
    <body>
        Exception is <%= exception %> generated...<br>
        Exception message is <%= exception.getMessage () %><br>
        <%@ include file="copyright.html" %>
    </body>
</html>
```

Copyright.html:

```
<html>
    <br><br><br><em>All copy rights are reserved for kvr.com<em>
</html>
```

Develop JSP pages which will participate in session?

Answer:

web.xml:

```
<web-app>
</web-app>
```

First.jsp:

```
<%@ page session="true" %>
<html>
    <body>
        <form name="first" action="Second.jsp">
            Enter your name : <input type="text" name="first_name"><br>
            <input type="submit" value="Send">
        </form>
    </body>
</html>
```

Second.jsp:

```
<%@ page session="true" %>
<html>
    <body>
        <% String str=request.getParameter ("first_name"); %>
        Your name is <h4><%= str %></h4>
        <% session.setAttribute ("name",str); %>
        <form action="Third.jsp"><br>
            Send the request to next page<br>
            <input type="submit" value="Send">
        </form>
    </body>
</html>
```

Third.jsp:

```
<%@ page session="true" %>
<html>
    <body>
        Value in third page from session objects is
        <h4><%= session.getAttribute ("name") %></h4>
    </body>
</html>
```

Day - 48:

2. Include directives:

Include is the directive to include the server side resource. The server side resource can be either an html file or JSP or a servlet.

If we include html file, it will be executed by **browser** when the response is rendering to the client. When we include a JSP or a servlet, it will be executed by **container**.

Syntax:

```
<% include file="file name to be included" %>
```

For example:

```
<% include file="copyright.html" %>
```

3. Taglib directives (Custom Tags):

The tags which are provided in JSP may not be solving the total problems of JSP programmer. In some situations, there is a possibility of repeating the same code many times by various JSP programmers and it leads to poor performance of a JSP application. In order to avoid the repetition of the code, it is highly desirable to develop the code **only one** time in the form of tags and use these tags by other JSP programmers. These tags are known as custom tags.

Advantages of custom tags:

1. Repetition of code (application logic) is reduced. Hence, we can achieve high performance and less storage cost.
2. We can achieve the slogan of WORA.
3. Custom tags provide simplicity for the JSP programmer in developing the application logic.

Steps for developing custom tags:

1. Decide which tag to be used along with prefix or short name, tag name and attribute names if required.
2. While we are choosing prefix it should not belongs to JSP, javax, javax and java.
3. After developing a custom tag one must specify the details about tag in a predefined file called **tld** (Tag Library Descriptor) file.
4. tld file contains declarative details about custom tags.
5. After developing tld file keep it into either WEB-INF folder directly or keep it into a separate folder called tlds folder and it in turns present into WEB-INF.
6. Whenever we make a request to a JSP page where we are using custom tag will give location of tld file.
7. The tld file gives information about tag handler class (JavaBeans class) in which we develop the arithmetic logic or business logic for the custom tag.

Syntax for specifying the location of tld files:

```
<% taglib uri="location of tld file"
        prefix="prefix or short name of custom tag" %>
```

For example:

```
<% taglib uri="/WEB-INF/tlds/x.tld" prefix="database" %>
```

Here, taglib is a directive used for given information regarding tld file and prefix or short name of custom tag.

Entries in tld file:

Every **tld file** gives declarative details about custom tags. The following structure gives information regarding prefix name, tag name, tag handler class name, attribute names, etc.

x.tld:

```
<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>database</shortname>
```

Day - 49:

Syntax:

If we pass the data at runtime to the `rtexprvalue` attribute, this attribute must be `true` otherwise it is false.

1. All the attributes of custom tag must be used as properties or data members in tag handler class in the same order in which they appear.
2. Every tag handler class must extend a predefined class called `javax.servlet.jsp.tagext.TagSupport`
3. `TagSupport` class contains the following two life cycle methods (which are automatically called by the JSP container) and they must be overridden in tag handler class.

Page 107

The legal values which are returned by doStartTag are:

```
public static final int SKIP_BODY;
public static final int EVAL_BODY_INCLUDE; }
```

Present in TagSupport class

doStartTag () method will be called by JSP container when starting of custom tag taken place <x:hello/>. SKIP_BODY is the constant which will be returned by doStartTag () method when we don't want to execute body of the custom tag. EVAL_BODY_INCLUDE is the constant which will be returned by doStartTag () method when we want to evaluate body of the custom tag.

The legal values which are returned by doEndTag () are:

```
public static final int EVAL_PAGE;
public static final int SKIP_PAGE; }
```

Present in TagSupport class

doEndTag () method will be called by JSP container when the closing tag of custom tag taken place <x:hello/>. EVAL_PAGE is the constant to be returned by doEndTag () method when we want to execute rest of the JSP page. SKIP_PAGE is the constant to be returned by doEndTag () method when we don't want to execute rest of the JSP page.

URI → Location of file which is not running

URL → Location of file which is running

Day - 50:

Develop a JSP page which illustrate the concept of TagSupport class, mean while get connection with database using custom tag?

Answer:

web.xml:

```
<web-app>
</web-app>
```

JdbcTag.jsp:

```
<html>
  <body>
    <%@ taglib prefix="table" uri="/WEB-INF/tlds/JdbcTag.tld" %>
    <center>
      <table:show username="scott" password="tiger" dsn="oradsn" table="product">
    </center>
  </body>
</html>
```

JdbcTag.java:

```
package tagpack;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
import java.io.*;
```

```
public class JdbcTag extends TagSupport
{
    String table, username, password, dsn;
    public void setUsername (String username)
    {
        this.username=username;
    }
    public void setPassword (String password)
    {
        this.password=password;
    }
    public void setDsn (String dsn)
    {
        this.dsn=dsn;
    }
    public void setTable (String table)
    {
        this.table=table;
    }
    Connection con=null;
    PreparedStatement ps=null;
    ResultSet rs=null;
    ResultSetMetaData rsmd=null;
    public int doStartTag () throws JspException
    {
        try
        {
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection ("jdbc:odbc:"+dsn,username,password);
            ps=con.prepareStatement ("select * from "+table);
            return EVAL_BODY_INCLUDE;
        }
        catch (Exception e)
        {
            throw new JspException (e.getMessage ());
        }
    }
    public int doEndTag () throws JspException
    {
        int i=1;
        try
        {
            rs=ps.executeQuery ();
            rsmd=rs.getMetaData ();
            pageContext.getOut ().print ("<table border=1><tr>");
            for (i=1; i<=rsmd.getColumnCount (); i++)
            {
                pageContext.getOut ().print ("<th>"+rsmd.getColumnName (i)+"</th>");
            }
            pageContext.getOut ().print ("</tr>");
            while (rs.next ())
            {
                pageContext.getOut ().print ("<tr>");
                for (i=1; i<=rsmd.getColumnCount (); i++)
                {
                    pageContext.getOut ().print ("<td>"+rs.getString (i)+"</td>");
                }
            }
        }
    }
}
```

```
        }
        pageContext.getOut ().print ("</tr>");
    }
    pageContext.getOut ().print ("</table>");
    return EVAL_PAGE;
}
catch (Exception e)
{
    throw new JspException (e.getMessage ());
}
};
```

JdbcTag.tld:

```
<taglib>
  <shortname>table</shortname>
  <tag>
    <name>show</name>
    <tagclass>tagpack.JdbcTag</tagclass>
    <bodycontent>empty</bodycontent>
    <attribute>
      <name>username</name>
      <required>true</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>password</name>
      <required>true</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>dsn</name>
      <required>true</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>table</name>
      <required>true</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

NOTE: We should write the attributes same in the order of jsp program we used.

Day -51:

Develop a JSP page which illustrate the concept of TagSupport class, mean while print the typed name on the browser using custom tag?

Answer:**web.xml:**

```
<web-app>
</web-app>
```

HelloTag.jsp:

```
<%@
    taglib uri="/WEB-INF/tlds/hello.tld" prefix="test"
%>
<html>
    <h3>This is example on custom tag</h3>
    <h4><test:hello name="Hyderabad"/></h4>
</html>
```

HelloTag.java (Tag Handler Class):

```
package t1;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class HelloTag extends TagSupport
{
    private String name;
    public void setName (String name)
    {
        this.name=name;
    }
    public int doEndTag () throws JspException
    {
        try
        {
            pageContext.getOut ().print ("Hello..! "+name);
        }
        catch (Exception e)
        {
            throw new JspException (e.getMessage ());
        }
        return EVAL_PAGE;
    }
};
```

Hello.tld:

```
<taglib>
    <shortname>test</shortname>
    <tag>
        <name>hello</name>
        <tagclass>t1.HelloTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute>
            <name>name</name>
            <required>true</required>
            <rtexprvalue>>false</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

NOTE:

pageContext is an object of javax.servlet.jsp.PageContext interface and it will be created automatically by JSP container. pageContext object is pre-declared in TagSupport class and this object will be available to each and every sub-class of TagSupport class.

STANDARD ACTIONS:

These are basically used to pass **runtime information to the container**. As a part of JSP we have the following *standard actions*; they are `<JSP:forward/>`, `<JSP:include/>`, `<JSP:param/>`, `<JSP:useBean/>`, `<JSP:setProperty/>` and `<JSP:getProperty/>`.

1. `<JSP:forward/>`:

When we want to forward a request and response to the destination JSP page from the source JSP we must use `<JSP:forward>`.

Syntax:**Without body:**

```
<JSP:forward page="relative or absolute path of JSP page"/>
```

With body:

```
<JSP:forward page=" relative or absolute path of JSP page">
    <JSP:param name="param name1" value="param value1"/>
    <JSP:param name="param name2" value="param value2"/>
</JSP:forward>
```

For example:

```
<JSP:forward page="y.jsp">
    <JSP:param name="v1" value="10"/>
    <JSP:param name="v2" value="20"/>
</JSP:forward>
```

When we use this tag we get the response of destination JSP page only but not source JSP page.

2. `<JSP:include/>`:

This tag is used for processing a client request by a source JSP page by including other JSP pages and static resources like html's. One source JSP can include 'n' number of server side resources and finally we get the response of source JSP only.

Syntax:**Without body:**

```
<JSP:include page="relative or absolute path of JSP page"/>
```

With body:

```
<JSP:include page=" relative or absolute path of JSP page">
    <JSP:param name="param name1" value="param value1"/>
    <JSP:param name="param name2" value="param value2"/>
</JSP:include>
```

For example-1:

```
<JSP:include page="y.jsp">
    <JSP:param name="v1" value="10"/>
    <JSP:param name="v2" value="20"/>
</JSP:include>
```

For example-2:

```
<JSP:include page="z.jsp">
    <JSP:param name="v3" value="30"/>
    <JSP:param name="v4" value="40"/>
</JSP:include>
```

3. <JSP:param/>:

This tag is used for passing the local data of one JSP page to another JSP page in the form of (key, value) pair.

Syntax:

```
<JSP:param name="name of the parameter" value="value of the parameter"/>
```

Here, name represents name of the parameter or attribute and it must be unique value represents value of the parameter and should be always string. <JSP:param/> tag should be used in connection with either <JSP:forward/> or <JSP:include/>.

For example-1:

```
<JSP:forward page="y.jsp">
    <JSP:param name="v1" value="10"/>
    <JSP:param name="v2" value="20"/>
</JSP:forward>
```

For example-2:

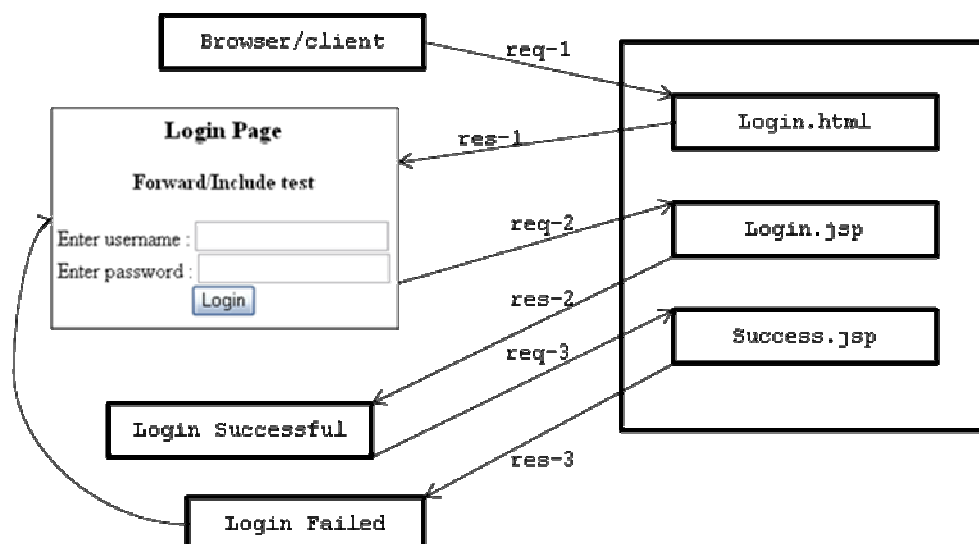
```
<JSP:include page="y.jsp">
    <JSP:param name="v1" value="10"/>
    <JSP:param name="v2" value="20"/>
</JSP:include>
```

For example-2:

```
<JSP:include page="z.jsp">
    <JSP:param name="v3" value="30"/>
    <JSP:param name="v4" value="40"/>
</JSP:include>
```

Day - 52:

Write a JSP page which illustrates the concept of <JSP:forward/> and <JSP:include/>?

**Answer:****web.xml:**

```
<web-app>
</web-app>
```



```
PreparedStatement ps=null;
public void jspInit ()
{
    try
    {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        con=DriverManager.getConnection      ("jdbc:oracle:thin:@localhost:1521:Hanuman",
"scott","tiger");
        ps=con.prepareStatement ("insert into student values (?, ?, ?, ?, ?)");
    }
    catch (Exception e)
    {
        out.println (e);
    }
} %>
<%
String stno=request.getParameter ("first_stno");
String name=request.getParameter ("first_stname");
String college=request.getParameter ("second_cname");
String marks1=request.getParameter ("second_marks");
String dob1=request.getParameter ("second_dob");
int number=Integer.parseInt (stno);
float marks=Float.parseFloat (marks1);
Date dob=Date.parseDate (dob1);
%>
<%!
try
{
    ps.setInt (1,number);
    ps.setString (2, name);
    ps.setString (3, college);
    ps.setFloat (4, marks);
    ps.setDate (5, dob);
    int i=ps.executeUpdate ();
    if (i>0)
    {
        %>
        <html>
            <body bgcolor="lightblue">
                <center>
                    <h3>Inserted Successfully</h3>
                </center>
                <a href="login.html">One more</a>
            </body>
        </html>
    }
    else
    {
        %>
        <html>
            <body bgcolor="lightblue">
                <center>
                    <h3>Try again</h3>
                </center>
                <a href="login.html">Home</a>
            </body>
        </html>
    }
}
catch (exception e)
```

```
        {  
            out.println (e);  
        }  
        %>  
</html>
```

JavaBeans in JSP

A JavaBeans class is a software reusable component. Every JavaBeans class must belong to a package. Since, it is reusable. Every JavaBeans class modifier must be public. Every JavaBeans class must contain set of data members (known as properties).

For each and every data member of JavaBeans class we must have set of set methods whose general representation is:

```
public void setXxx (datatype FormalVariableName)  
{  
    .....;   
    .....;   
    .....;   
}
```

For each and every data member of JavaBeans class we must have set of get methods whose general representation is:

```
public datatype getXxx ()  
{  
    .....;   
    .....;   
    .....;   
}
```

The set of set methods are used for setting the values to JavaBeans class object whereas set of get methods are used for getting the values from JavaBeans class object.

Properties or characteristics of JavaBeans:

Every JavaBeans class contains **simple property**, **boolean property** and **indexed properties**.

- A **simple property** is one in which a method takes and returns elementary or single value (set of set and get methods are known as *simple properties* of a JavaBeans class.)
- A **boolean property** is one in which a method takes or return boolean value.
- An **indexed property** is one in which a method takes or return array of values.

Develop a JavaBeans class which will check whether the username and password correct or not?

Answer:

Test.java:

```
package abc;  
public class Test  
{  
    String uname;  
    String pwd;  
    public void setUsername (String uname)  
    {  
        this.uname=uname;  
    }  
    public void setPassword (String pwd)
```

```
{
    this.pwd=pwd;
}
public String getUname ()
{
    return (uname);
}
public String getPwd ()
{
    return (pwd);
}
public boolean validate ()
{
    if (uname.equals ("kalpana") && pwd.equals ("test"))
    {
        return (true);
    }
    else
    {
        return (false);
    }
}
};
```

Day - 53:

NOTE: It is highly recommended to use a JSP page with scriptless.

Using a JavaBean class into JSP page the following tags are used to use a JavaBean class as a part of JSP:

4. <JSP:useBean/>: This tag is used for creating an object of JavaBeans class as a part of JSP page.

Syntax:

```
<JSP:useBean    id="object name of a JavaBeans class"
                class="fully qualified name of JavaBeans class"
                scope="scope attribute"
                type="name of base interface or class" />
```

- Here, *JSP* represents prefix or short name of *useBean* tag.
- *useBean* represents a tag for representing details about JavaBeans class.
- *id* and *name* are the mandatory attributes of *useBean* tag.
- *id* represents object name of JavaBeans class.
- *name* represents fully qualified name of JavaBeans class.
- *scope* represents the visibility or accessibility of a JavaBeans class.

The **scope** attribute represents any one of the following:

- **page** - represents a JavaBeans class object can be accessed in current JSP page only. It cannot be accessed in another JSP pages. By default the scope is *page*.

- **request** - represents a JavaBeans class object can be accessed in those JSP pages which are participating in processing a single request.
- **session** - represents a JavaBeans class object can be accessed in all the JSP pages which are participating in a session and it is not possible to access in those JSP pages which are not participating in session.
- **application** - represents a JavaBeans class object can be accessed in all the JSP pages which belongs to the same web application but it is not possible to access in those JSP pages which are belongs to other web applications.

The **type** attribute represents specification of base interface or class name of a JavaBeans class.

For example:

```
<JSP:useBean id="eo"
              class="ep.Emp"
              scope="session"
              type="ep.GenEmp" />
```

When the above statement is executed the container creates an object eo is created in the following way:

```
ep.GenEmp eo=new ep.Emp ();
```

If we are not specifying the value for type attribute then the object eo is created in the following way:

```
ep.Emp eo=new ep.Emp ();
```

NOTE:

In the above `<JSP:useBean/>` tag if we use a tag called `<JSP:setProperty/>` then that tag becomes body tag of `<JSP:useBean/>` tag.

5. `<JSP:setProperty/>`:

This tag is used for setting the values to the JavaBeans class object created with respect to `<JSP:useBean/>` tag.

Syntax-1:

```
<JSP:setProperty      name="object name of a JavaBeans class"
                      Property="property name of JavaBeans class"
                      Value="value for property" />
```

For example:

```
<JSP:useBean id="eo" class="ep.Emp">
    <JSP:setProperty  name="eo"
                      Property="empno"
                      Value="123" />
</JSP:useBean>
```

When the above statement is executed by the container, the following statement will be taken place.

```
ep.Emp eo=new ep.Emp ();
eo.setEmpno ("123");
```

The above syntax used to call a specific set method by passing a specific value statically.

Syntax-2:

```
<JSP:setProperty      name="object name of a JavaBeans class"
                      property="*" />
```

The above syntax is used for calling all generic set methods by passing there values dynamically.

For example:

```
<JSP:useBean id="eo" class="ep.Emp">
    <JSP:setProperty name="eo" property="*" />
</JSP:useBean>
```

Dynamically we can pass the values through HTML program to a JSP page. All the form fields of HTML program must be similar to data members or properties of a JavaBeans class and in the same order we must define set of set methods.

Day - 54:

6. <JSP:getProperty/>: This tag is used for retrieving the values from JavaBeans class object.

Syntax:

```
<JSP:getProperty      name="object name of JavaBeans class"
                      property="property name of JavaBeans class" />
```

For example:

```
<JSP:getProperty name="eo" property="empno" />
[or]
<%= eo.getEmpno () %>
```

For example (bean1):**web.xml:**

```
<web-app>
</web-app>
```

Bean.html:

```
<html>
<body>
    <h3>Bean tag test</h3>
    <form name="b1" action="bean.jsp" method="post">
        Enter ur name : <input type="text" name="b1_name"><p>
        Select the language :&nbsp;
        <select name="b1_lang">
            <option value=""></option>
            <option value="c"> C </option>
            <option value="c++"> C++ </option>
            <option value="java"> Java </option>
            <option value=".net"> .NET </option>
        </select><p>
        <input type="submit" value="Send">&nbsp;
```

```
        <input type="reset" value="Clear">
    </form>
</body>
</html>
```

Bean.jsp:

```
<html>
<body>
    <jsp:useBean id="obj" class="tp.TechBean">
        <jsp:setProperty name="obj" property="*" />
    </jsp:useBean>
    <h3>Result of bean action tags</h3>
    Hello <jsp:getProperty name="obj" property="bl_name" /><p>
    <jsp:getProperty name="obj" property="bl_lang" /><p>
    <jsp:getProperty name="obj" property="langComments" /><p>
    <h3>Result of expression tags</h3>
    Name : <%= obj.getName () %><br>
    Language : <%= obj.getLang () %><br>
    Comment : <%= obj.getLangComments %>
</body>
</html>
```

TechBean.java:

```
package tp;
public class TechBean
{
    String name;
    String lang;
    public TechBean () //recommended to write
    {
    }
    public void setName (String name)
    {
        this.name=name;
    }
    public void setLang (String lang)
    {
        this.lang=lang;
    }
    public String getName ()
    {
        return name;
    }
    public String getLang ()
    {
        return lang;
    }
    public String getLangComments ()
    {
        if (lang.equals ("c"))
        {
            return ("Mr. Kalyan Reddy is the best faculty in Hyderabad");
        }
        else if (lang.equals ("c++"))
        {
        }
    }
}
```

```
        return ("Kalyan IT is the best institute for it");
    }
    else if (lang.equals ("java"))
    {
        return ("Mr KVR is the best faculty in Hyderabad");
    }
    else if (lang.equals (".net"))
    {
        return ("Mr. Nageswara is the best faculty in Hyderabad");
    }
    else
    {
        return ("No idea...!");
    }
}
};
```

For example (bean2):**web.xml:**

```
<web-app>
</web-app>
```

CheckBean.html:

```
<html>
<body>
    <form name="checkbean" action="CheckBean.jsp" method="post">
        Enter user name : <input type="text" name="checkbean_name" value=""><br>
        Enter password : <input type="password" name="checkbean_pwd" value=""><br>
        <input type="submit" value="Send">&nbsp;
        <input type="reset" value="Clear">
    </form>
</body>
</html>
```

CheckBean.jsp:

```
<%@ page import="mypack.CheckBean" %>
<jsp:useBean id="check" class="CheckBean" scope="session">
    <jsp:setProperty name="check" property="*" />
</jsp:useBean>
<%= check.validate () %>
```

CheckBean.java:

```
package mypack;
public class CheckBean
{
    String uname;
    String pwd;
    public CheckBean ()
    {
    }
    public void setUsername (String uname)
    {
        this.uname=uname;
    }
}
```

```
public void setPwd (String pwd)
{
    this.pwd=pwd;
}
public String getUname ()
{
    return uname;
}
public String getPwd ()
{
    return pwd;
}
public boolean validate ()
{
    if (uname.equals ("asha") && pwd.equals ("krishna"))
    {
        return (true);
    }
    else
    {
        return (false);
    }
}
};
```

Day - 54:

Develop the JSP pages which illustrate the concept of implicit application object (application is an implicit object created with respect to ServletContext and the data can be accessed through out the entire web application)?

Answer:**First.jsp:**

```
<html>
    <body>
        <h3>Application variable is defining</h3>
        <%!
            String name="asha";
            String pwd="krishna";
        %>
        <%
            application.setAttribute ("val1",name);
            application.setAttribute ("val2",pwd);
        %>
        <a href="http://localhost:7001/application/second.jsp">Click here</a>
    </body>
</html>
```

Second.jsp:

```
<html>
    <body>
        <h3>Application variable is retrieving</h3>
        <%!
```

```
String name;
String pwd;

%>
<%
    name=(String)application.getAttribute ("val1");
    pwd=(String)application.getAttribute ("val2");

%>
Name :: <%= name %>
Password :: <%= pwd %>
</body>
</html>
```

Day - 55:

JSTL (JSP Standard Template Library)

In industry developing custom tags will take lot of time by the java programmer. In order to minimize this application development time, various server vendors came forward and developed there own tags from the fundamental tags to xml, internationalization tags (i18n → 18 mean languages). But all these tags are server dependent.

In later stages SUN micro system has collected all these tags which are developed by various server vendors and converted into server independent with the help of JavaSoft Inc., USA.

JSTL contains four tags; they are **core tags**, **database tags**, **formatting tags** (internationalization [i18n]) and **xml tags** [Only core tags and database tags are related to advanced java].

- In order to deal with JSTL, one must deal with two jar files , they are **jstl.jar** and **standard.jar**
- These jar files contains the tag handler class information regarding the tags which are available in JSTL.

SUN micro system has given a fixed uri, tld file and prefix name for various tags and they are as follows:

tag name	URI	tld file	prefix
Core	http://java.sun.com/jstl/core	c.tld	c
Database	http://java.sun.com/jstl/sql	sql.tld	sql
Formatting	http://java.sun.com/jstl/fmt	fmt.tld	fmt
Xml	http://java.sun.com/jstl/xml	xml.tld	xml

NOTE:

The two standard jar files (jstl.jar and standard.jar) will be available in Tomcat 5.0\webapps\jsp-examples\WEB-INF\lib

When we use any JSTL tags as a part of our JSP program we must first configure the web.xml file and the sample entries in web.xml is shown below:

web.xml:

```
<web-app>
    <taglib>
        <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
```

```
        <taglib-location>/WEB-INF/tlds/c.tld</taglib-location>
    </taglib>
</web-app>
```

1. **Core tags:**

In core tags, we have < c:set.... />, < c:out.... />, < c:remove.... />, < c:if.... />, < c:forEach.... /> and < c:choose.... />.

Day - 56:

< c:set.... /> Syntax:

```
< c:set      var="variable name"
              value="value of the variable"
              scope="scope of the variable" >
```

For example:

```
< c:set var="a" value="10" scope="request">
```

When this statement is executed the JSP container will prepare the following statement:

```
request.setAttribute ("a", 10);
```

< c:out.... /> Syntax:

This tag is used for printing the values on the browser when the client makes a request.

```
< c:out value="$ {variable name or expression}" />
```

For example:

```
< c:set var="a" value="10" />
< c:set var="b" value="20" />
< c:out value="$ {a}" /> → 10
< c:out value="$ {b}" /> → 20
< c:out value="$ {a+b}" /> → 30
```

< c:remove.... /> Syntax:

This tag is used for removing an attribute from scope object [request, session and application but not page].

```
< c:remove var="variable name" />
```

For example:

```
< c:set var="a" value="10" scope="request" />
< c:set var="b" value="20" scope="request"/>
< c:remove var="a" /> → request.removeAttribute (a)
```

< c:if.... /> Syntax:

```
< c:if test="$ {test condition}">
Block of statements;
</ c:if >
```

For example:

```
< c:set var="a" value="20" />
< c:set var="b" value="30" />
< c:if test="$ {a>b}" >
<h3>a is greater than b</h3>
```

```
</ c:if >
< c:if test="$ {b>a}" >
<h3>b is greater than a</h3>
</ c:if >
```

```
< c:forEach.... /> Syntax:
< c:forEach var="name of the variable"
            begin="starting value"
            end="ending value"
            step="updation" >
Block of statements;
</ c:forEach >
```

```
< c:choose.... /> Syntax:
< c:choose >
    < c:when test="$ {test condition 1}">
        Block of statements;
    </ c:when>
    < c:when test="$ {test condition 1}">
        Block of statements;
    </ c:when>
    .....
    < c:otherwise >
        Block of statements;
    </ c:otherwise >
</ c:choose >
```

In order to take the html form data into a jstl program one can use the following syntax:

param.html form field name

Here, *param* is acting as an implicit object of HttpServletRequest object.

For example:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
    <body>
        <c:choose>
            <c:when test="${Param.p>0}">p is positive</c:when>
            <c:when test="${Param.p<0}">p is negative</c:when>
            <c:otherwise>p is zero</c:otherwise>
        </c:choose>
    </body>
</html>
```

To run: <http://localhost:7001/jstl/ex2.jsp?p=10>

In order to get wheather the html form field name is containing a value or not in JSTL we use the following statement:

empty param.http form field name

For example:

```
<c:if test="${emptyParam.p}">
```

It returns true when p does not contain a value.

For example:

```
<%@ taglib uri=" http://java.sun.com/jstl/core " prefix="c" %>
```



```
<html>
  <body>
    <c:set var="msg" value="welcome"/>
    <c:if test="${!empty Param.uname}">
      <c:out value="${msg}"/>
      <c:out value="${Param.uname}"/>
    </c:if>
  </body>
</html>
```

Day - 57:

< c:forTokens.... /> Syntax:

```
< c:forTokens      var="name of the variable"
                  items="list of string values"
                  delims="delimiter which is separating string value" >
Block of statements;
</ c:forTokens >
```

For example:

```
<c:forTokens var="x" items="{abc, pqr, klm}" delims=",">
  <c:out value="${x}"/>
</c:forTokens>
```

[or]

```
<c:set name="items" value="{abc, pqr, klm}"/>
```

In java, String items []={ "abc", "pqr", "klm" }

```
<c:forTokens var="stname" items="${items}">
  <c:out value="${stname}"/>
</c:forTokens>
```

Write a JSTL program which illustrates the concept of < c:forTokens />?

Answer:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <body>
    List of Students<br>
    <c:set var="str" value="abc, pqr, aaa" />
    <c:forTokens var="sname" items="${str}" delims=",">
      <c:out value="${sname}" /><br>
    </c:forTokens>
  </body>
</html>
```

2. Database tags:

In order to deal with database through JSTL we must use the following URI and prefix:

URI=" <http://java.sun.com/jstl/sql> " prefix="sql"

The above two details must be specified through a directive called taglib and we must declare them in web.xml

Since, we are dealing with database we must have the appropriate jar files i.e., *classes111.jar* for oracle database.

In order to deal with database in JSTL we must use the tags `< sql:setDataSource.... />` and `< sql:query.... />`.

`< sql:setDataSource.... />`:

This tag is used for specifying how to load database drivers and obtain the connection by specifying data source name, user name and password if required.

Syntax:

```
< sql:setDataSource      var="data source name"
                        driver="name of the driver"
                        url="type of the driver"
                        username="user name"
                        password="name of the password" >
```

`< sql:query.... />`:

This tag is used for passing the query to perform insertion, deletion, updation and selection.

Syntax:

```
< sql:query  var="variable name"
             datasource="${data source name}"
             sql="name of the query" />
```

Write a JSTL program which retrieves the data from the database?

Answer:

```
<html>
  <body>
    <%@ page import="java.sql.*" %>
    <%@ page import="oracle.jdbc.driver.*" %>
    <%
      try
      {
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
        out.print ("Drivers loaded");
        Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@
localhost:1521:BudDinu","scott","tiger");
        out.print ("Connection established");
        String url="select * from emp";
        PreparedStatement ps=con.prepareStatement (url);
        ResultSet rs=ps.executeQuery ();
        ResultSetMetaData rsmd=rs.getMetaData ();
        ColCount=rsmd.getColumnCount ();
        i=1;

        %>
        <table bgcolor=lightyellow align=center border=2>
        <%
          while (i<=ColCount)
          {
            %>
            <th>
            <%          out.print (rsmd.getColumnName (i));          %>
            </th>
            <%
              i++;
```

```
        }
        while (rs.next ())
        {
            %>
            <tr>
            <%
                j=1;
                while (j<=ColCount)
                {
                    %>
                    <td>
                    <%
                        out.print (rs.getString (j));          %>
                    </td>
                    <%
                        j++;
                    %>
                }
            </tr>
            <%
                rs.close ();
                con.close ();
            %>
        }
        catch (Exception e)
        {
            e.printStackTrace ();
        }
    %>
</body>
</html>
```

Day - 58:**SWINGS**

- In the earlier days SUN micro system; we have a concept called awt.
- awt is used for creating GUI components.
- All awt components are written in 'C' language and those components appearance is changing from one operating system to another operating system. Since, 'C' language is the platform dependent language.
- In later stages SUN micro system has developed a concept called swings.
- Swings are used for developing GUI components and all swing components are developed in java language.
- Swing components never change their appearance from one operating system to another operating system. Since, they have developed in platform independent language.

Differences between awt and swings:

Awt	Swings
1. awt components are developed in 'C' language.	1. Swing components are developed in java language.
2. All awt components are platform dependent.	2. All swing components are platform independent.

3. All awt components are heavy weight components, since whose processing time and main memory space is more.	3. All swing components are light weight components, since its processing time and main memory space is less.
---	---

NOTE: All swing components in java are preceded with a letter 'J'.

For example:

```
JButton JB=new JButton ("ok");
```

All components of swings are treated as classes and they are belongs to a package called `javax.swing.*`

In swings, we have two types of components. They are **auxiliary components** and **logical components**.

- *Auxiliary components* are those which we can touch and feel. For example, mouse, keyboard, etc.
- *Logical components* are those which we can feel only.

Logical components are divided into two types. They are **passive or inactive components** and **active or interactive components**.

- *Passive components* are those where there is no interaction from user. For example, JLabel.
 - *Active components* are those where there is user interaction. For example, JButton, JCheckbox, JRadioButton, etc.
- ✓ In order to provide functionality or behavior to swing GUI active components one must import a package called `java.awt.event.*`
 - ✓ This package contains various classes and interfaces which provides functionality to active components.
 - ✓ EDM is one which always provides the functionality to GUI active components.

Steps in EDM:

1. Every GUI active component can be processed in two ways. They are **based on name or label of the component** and **based on reference of the component**. Whenever we interact with any GUI component whose reference and label will be stored in one of the predefined class object whose general notation is `xxxEvent` class.

For example:

```
JButton → ActionEvent
```

```
JCheckbox → ItemEvent
```

2. In order to provide behavior of the GUI component we must write some statements in methods only. And these methods are given by SUN micro system without definition. Such type of methods is known as abstract methods. In general, all abstract methods present in interfaces and those interfaces in swings known as Listeners. Hence, each and every interactive component must have the appropriate Listener whose general notation is `xxxListener`.

For example:

```
JButton → ActionListener
```

```
JCheckbox → ItemListener
```

3. Identify the abstract methods which are present in xxxListener to provide functionality to GUI component by overriding the abstract method.

For example:

JButton ActionListener → public abstract void actionPerformed (ActionEvent)

JCheckbox ItemListener → public abstract void itemStateChanged (ItemEvent)

4. Every GUI interactive component must be registered with appropriate Listener. Each interactive component will have the following generalized method to register or unregister with appropriate Listener.

public void addXxxListener (XxxListener) → Registration

public void removeXxxListener (XxxListener) → Unregistration

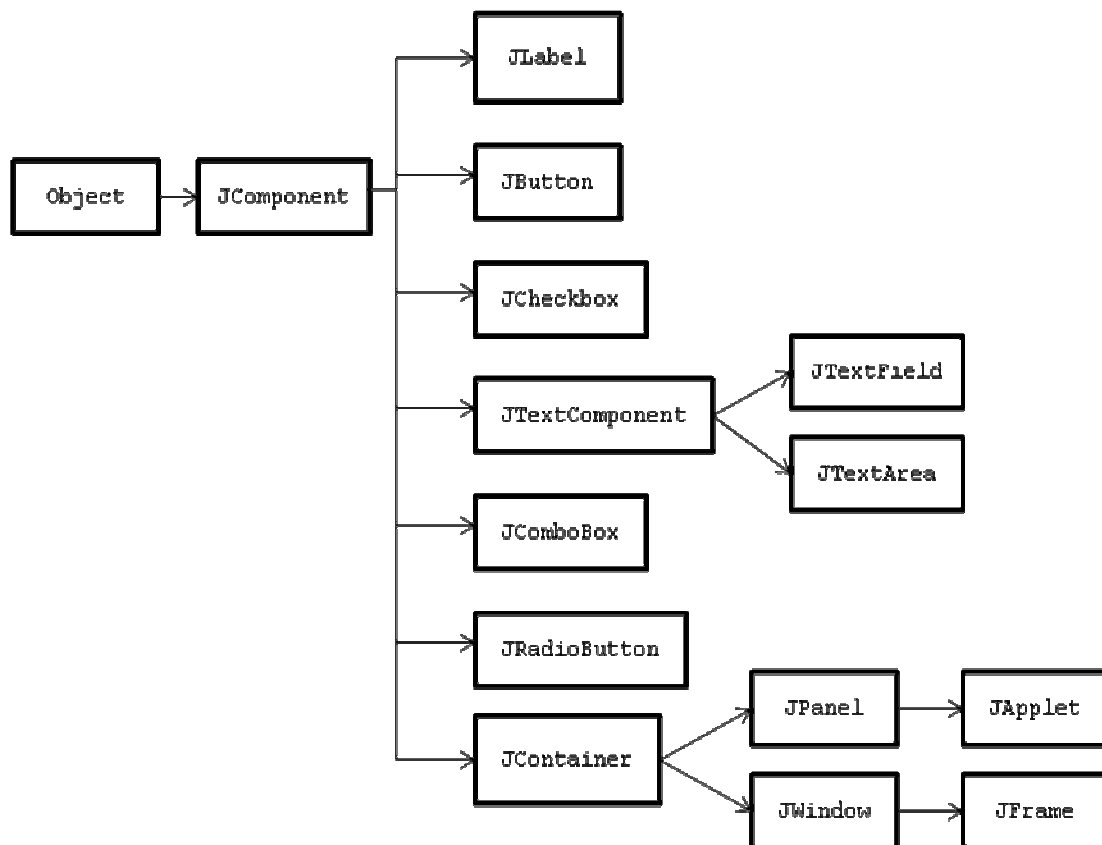
For example:

public void addActionListener (ActionListener) → Registration

public void removeActionListener (ActionListener) → Unregistration

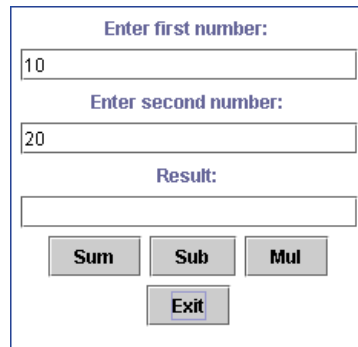
Day - 59:

Hierarchy chart in Swings:



NOTE: Creating any component is nothing but creating an object of appropriate swing component class.

Develop the following application:



Answer:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Op extends Frame implements ActionListener
{
    JLabel jl1, jl2, jl3;
    JTextField jtf1, jtf2, jtf3;
    JButton jb1, jb2, jb3, jb4;
    Op ()
    {
        setTitle ("Operations");
        setSize (200, 200);
        setLayout (new FlowLayout ());
        jl1=new JLabel ("Enter first number: ");
        jl2=new JLabel ("Enter second number: ");
        jl3=new JLabel ("Result: ");
        jtf1=new JTextField (20);
        jtf2=new JTextField (20);
        jtf3=new JTextField (20);
        jb1=new JButton ("Sum");
        jb2=new JButton ("Sub");
        jb3=new JButton ("Mul");
        jb4=new JButton ("Exit");
        add (jl1);add (jtf1);
        add (jl2);add (jtf2);
        add (jl3);add (jtf3);
        add (jb1);add (jb2);add (jb3);add (jb4);
        jb1.addActionListener (this);
        jb2.addActionListener (this);
        jb3.addActionListener (this);
        jb4.addActionListener (this);
        setVisible (true);
    }
    public void actionPerformed (ActionEvent ae)
    {
        if (ae.getSource ()==jb1)
        {
            String s1=jtf1.getText ();
            String s2=jtf2.getText ();
            int n3=Integer.parseInt (s1)+Integer.parseInt (s2);
```

```
        String s3=String.valueOf (n3);
        jtf3.setText (s3);
    }
    if (ae.getSource ()==jb2)
    {
        String s1=jtf1.getText ();
        String s2=jtf2.getText ();
        int n3=Integer.parseInt (s1)-Integer.parseInt (s2);
        String s3=String.valueOf (n3);
        jtf3.setText (s3);
    }
    if (ae.getSource ()==jb3)
    {
        String s1=jtf1.getText ();
        String s2=jtf2.getText ();
        int n3=Integer.parseInt (s1)*Integer.parseInt (s2);
        String s3=String.valueOf (n3);
        jtf3.setText (s3);
    }
    if (ae.getSource ()==jb4)
    {
        System.exit (0);
    }
}
};
class OpDemo
{
    public static void main (String [] args)
    {
        Op ol=new Op ();
    }
};
```

Day - 60:

FILTERS

A filter is a java program which will handle pre-requests and post-responses at server side. Filter programs always runs in the **background** only.

Advantages of filters:

1. It provides 100% security to the server side applications.
2. We can achieve data compression, data encryption, auditing and authentication.

It is highly recommended to develop the filter programs to deal with background related tasks such as checking username and password, validations, etc. In servlet, its always recommended to write the business logic by avoiding the background related tasks.

In order to develop any filter application we must deal with the following interfaces:

```
javax.servlet.Filter
javax.servlet.FilterConfig
javax.servlet.FilterChain
```

javax.servlet.Filter: Filter is an interface which provides the life cycle methods for developing filter applications. To develop any filter application the user defined class must implement javax.servlet.Filter (interface).

For example:

```
public class X implements javax.servlet.Filter
{
    .....;
    .....;
}
```

Life cycle methods of Filter: The Filter interface contains the following life cycle methods; they are:

```
public void init (FilterConfig);
public void doFilter (ServletRequest, ServletResponse, FilterChain);
public void destroy ();
```

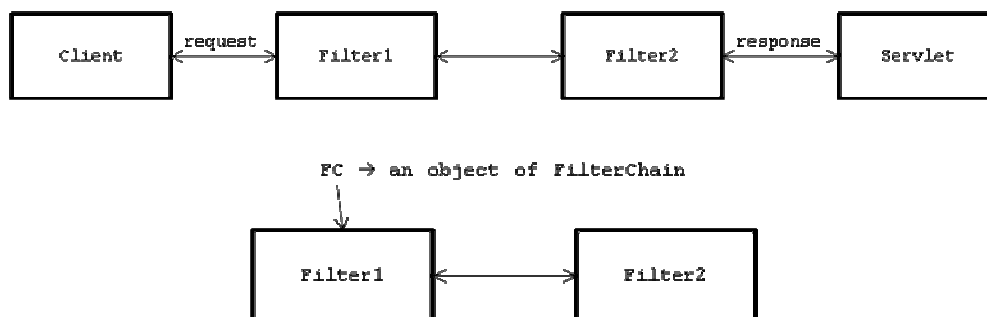
javax.servlet.FilterConfig:

FilterConfig is an interface which will be created by container and it contains initialization parameter details and technologies information for a filter and this information must be specified in web.xml. This object will exist one for filter in a web application.

javax.servlet.FilterChain:

FilterChain is an interface whose object created by container and it is pointing to group of filters (chain of filters) which are participating in pre-request process.

For example:



```
public void doFilter (ServletRequest req, ServletResponse res, FilterChain FC)
{
    .....;
    .....;
    FC.doFilter (req, res);
}
```

FC.doFilter (req, res) will call next filter which is available filter chain. In whichever order we write the declaration details of filter in web.xml in the same order filters will be called.

Filter entries related to web.xml:

web.xml:

```
<web-app>
    <filter>
```



```
<filter-name>abc</filter-name>
<filter-class>FilterEx</filter-class>
</filter>
<servlet>
    <servlet-name>pqr</servlet-name>
    <servlet-class>ServletEx</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>pqr</servlet-name>
    <url-pattern>/Serv</url-pattern>
</servlet-mapping>
<filter-mapping>
    <filter-name>abc</filter-name>
    <url-pattern>/Filt</url-pattern>
</filter-mapping>
</web-app>
```

Day - 61:

Develop a java program which will illustrate the concept of Filters?

Answer:**web.xml:**

```
<web-app>
    <filter>
        <filter-name>abc</filter-name>
        <filter-class>FilterEx</filter-class>
    </filter>
    <servlet>
        <servlet-name>pqr</servlet-name>
        <servlet-class>ServletEx</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>pqr</servlet-name>
        <url-pattern>/Serv</url-pattern>
    </servlet-mapping>
    <filter-mapping>
        <filter-name>abc</filter-name>
        <url-pattern>/Filt</url-pattern>
    </filter-mapping>
</web-app>
```

FilterEx.java:

```
import javax.servlet.*;
import java.io.*;
public class FilterEx implements Filter
{
    public FilterEx ()
    {
```

```
        System.out.println ("Inside constructor of Filter class");
    }
    public void init (FilterConfig fcon) throws ServletException
    {
        System.out.println ("Inside init () method of Filter class");
    }
    public void doFilter (ServletRequest req, ServletResponse res, FilterChain fc) throws IOException,
ServletException
    {
        System.out.println ("Inside doFilter () method of Filter class");
        int x=Integer.parseInt (req.getParameter ("first"));
        int y=Integer.parseInt (req.getParameter ("second"));
        if (x<0 || y<0)
        {
            PrintWriter pw=res.getWriter ();
            pw.print("<html><body>Sorry!! Ur input should be only positive numbers</body></html>");
        }
        else
        {
            fc.doFilter (req, res);
        }
    }
    public void destroy ()
    {
        System.out.println ("Inside destroy () method of Filter class");
    }
};
```

ServletEx.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ServletEx extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {
        System.out.println ("Inside doGet () method of Servlet class");
        int x=Integer.parseInt (req.getParameter ("first"));
        int y=Integer.parseInt (req.getParameter ("second"));
        int z=x+y;
        PrintWriter pw=res.getWriter ();
        pw.println ("<html><body>Result is "+z+"</body></html>");
    }
};
```

CONNECTION POOLING

Connection pooling is the process of group of readily available unnamed connections. As a java programmer we must use one of the name connections from connection pooling and use it in a java program.

In connection pooling, we can achieve concurrent execution and we can develop 3-tier applications. In connection pooling, we make use of Type-3 driver which are provided by server vendors.

Connection pooling is a unique concept provided by **weblogic** for achieving concurrent execution.

Day - 62:

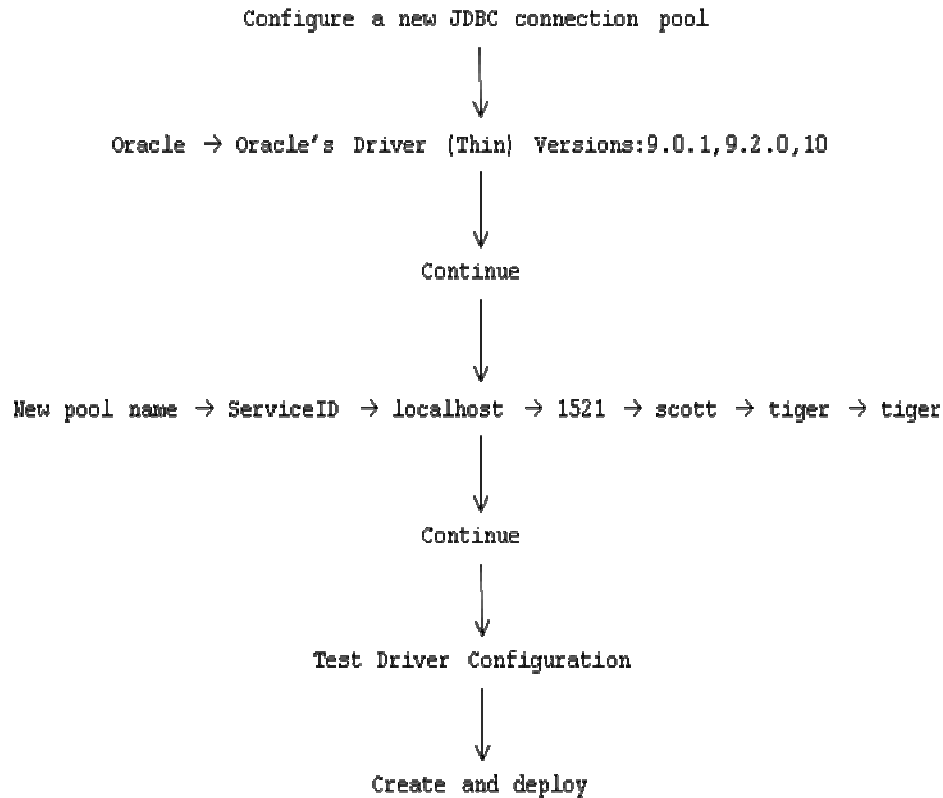
Steps for connection pooling: Connection pooling service provided by application servers only.

1. Go to weblogic console by opening a browser. <http://localhost:7001/console>
2. Enter username and password of weblogic.

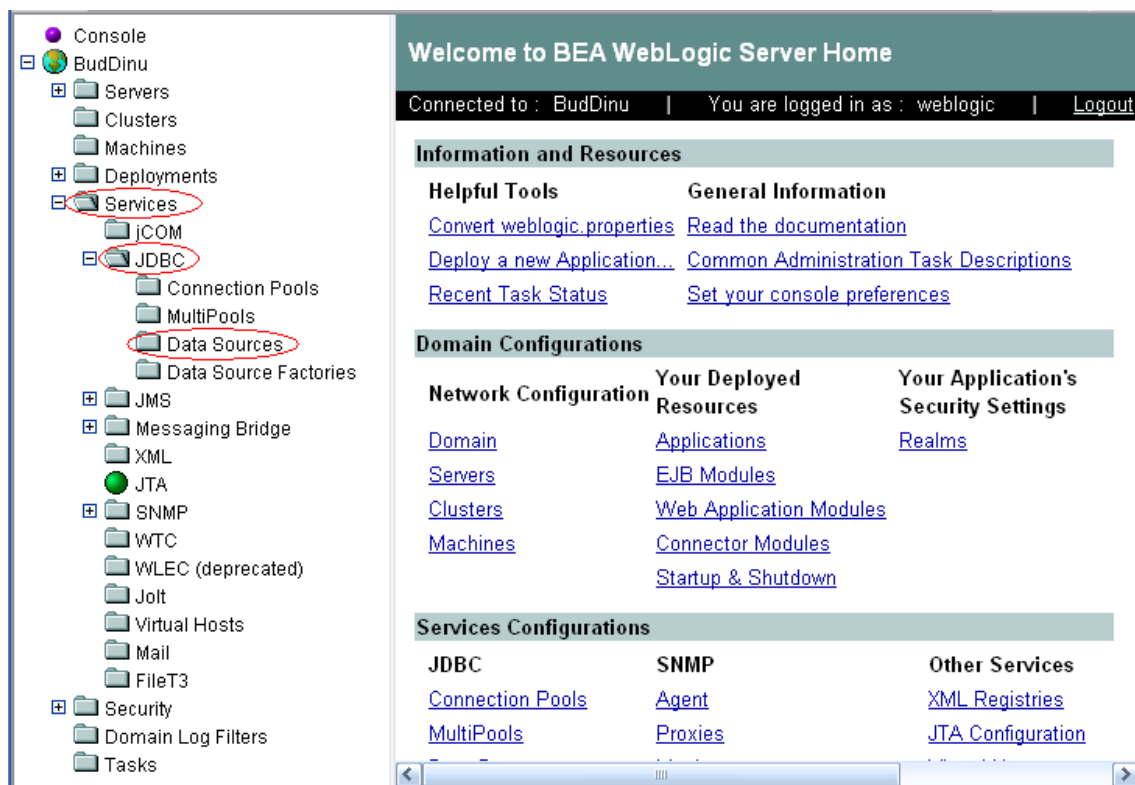


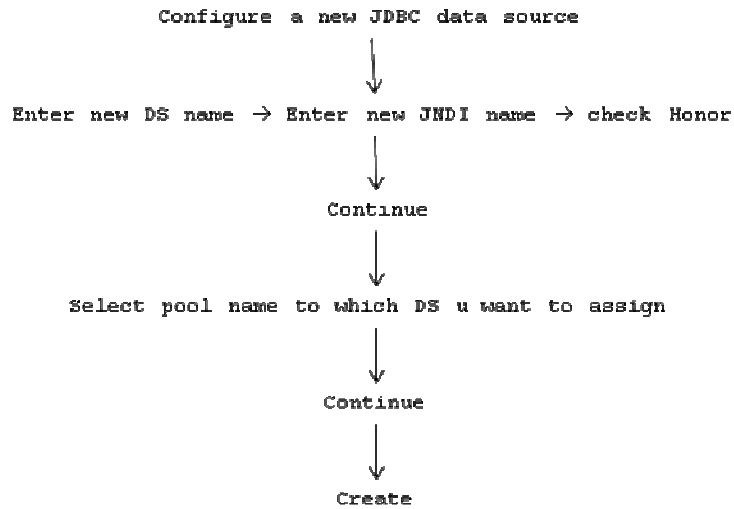
3. After entering username and password we will get the following window:





4. After choosing connection pool option follow the following steps:



Steps for creating data sources:**NOTE:**

In a Servlet program we should always use JNDI name which in turns pointing to appropriate data source name and it points to one of named connection in connection pool.

For example:**web.xml:**

```
<web-app>
    <servlet>
        <servlet-name>abc</servlet-name>
        <servlet-class>FirstConPoolServ</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>abc</servlet-name>
        <url-pattern>/ServPool</url-pattern>
    </servlet-mapping>
</web-app>
```

index.html:

```
<html>
    <body bgcolor=lightblue>
        <center>
            <form action= "./ServPool">
                Enter table name: <input type="text" name="table" value=""><br>
                <input type="submit" value="Bring data">
            </center>
        </body>
</html>
```

FirstConPoolServ.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.InitialContext;
public class FirstConPoolServ extends HttpServlet
```

```
{
    public void service (HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException
    {
        PrintWriter pw=res.getWriter ();
        String tabname=req.getParameter ("table");
        try
        {
            Connection con=getPoolCon (); // our own method
            Statement st=con.createStatement ();
            ResultSet rs=st.executeQuery ("select * from "+tabname);
            ResultSetMetaData rsmd=rs.getMetaData ();
            int ColCount=rsmd.getColumnCount ();
            pw.println ("<html><body bgcolor=#ffffa9><center><h3>Details of "+tabname);
            pw.println ("</h3><br><table><tr bgcolor=lightblue>");
            for (int col=1; col<=ColCount; col++)
            {
                pw.println ("<th>"+rsmd.getColumnLabel (col)+"&nbsp;&nbsp;&nbsp;</th>");
            }
            pw.println ("</tr>");
            while (rs.next())
            {
                pw.println ("<tr bgcolor=#ffffa9>");
                for (int i=1; i<=ColCount; i++)
                {
                    pw.println ("<td>"+rs.getString (i)+"&nbsp;&nbsp;&nbsp;</td>");
                }
                pw.println ("</tr>");
            }
            pw.println ("</table><br>To view another table<b><a href=index.html>Click here</a></b>");
            pw.println ("</center></body></html>");
        }
        catch (Exception e)
        {
            pw.println ("<html><body bgcolor=#ffffa9><center><h3>Table does not exist in database");
            pw.println ("</h3>To view another table<b><a href=index.html>Click here</a></b>");
            pw.println ("</center></body></html>");
            System.out.println (e);
        }
    }
    public Connection getPoolCon ()
    {
        Connection con=null;
        try
        {
            InitialContext ic=new InitialContext ();
            DataSource ds=(DataSource) ic.lookup ("FirstJNDI");
            con=ds.getConnection ();
        }
        catch (Exception e)
        {
            System.out.println (e);
        }
        return con;
    }
};
```