JDBC

JDBC

As if know it is known as trademark.
But earlier it is called as Java Database Connectivity.
RAM is a temporary storage device or medium.
During the program execution our data will store in RAM.
Once the program execution is completed we will loss the data.
To overcome this limitation we are making our application writing the data into file or database software.
Files and Database software's act like a permanent storage device or medium.

Persistence

The process of storing and managing the data for a long period of time is called persistence.

Important Terminologies

1) Persistence Store

It is a place where we can store and manage the data for a long period of time.
ex:

Database software's
Files

2) Persistence Data

Data of a persistence store is called persistence data.
ex:

tables / collections
records

3) Persistence operation

Insert, update, delete, create and etc are called persistence operation.
In the real-time this operation is also known as CURD operation, CRUD operation, SCUD operation.
ex:

C - create	S - select
U - update	C - create
R - read	U - update
D - delete	D - delete

4) Persistence logic

A logic which is capable to perform persistence operations is called persistence logic.
ex:

JDBC code
Hibernate code
IOStream

5) Persistence technology
A technology which is used to develop persistence logic is called persistence technology.
ex:
JDBC
Hibernate
JPA
and etc.

Q) What is JDBC?
JDBC is a persistence technology which is used to develop persistence logic having the capability to perform persistence operations on persistence data of a persistence store.

Note:
```
                    JDBC Code
JavaApp ------------------------------------- DB S/W's

                    IOStream
JavaApp ------------------------------------- Files
                Serialization/Deserialization
```

Serialization
The process of storing object data into a file is called serialization.
In serialization, object will not store in a file instead object data will store in a file.

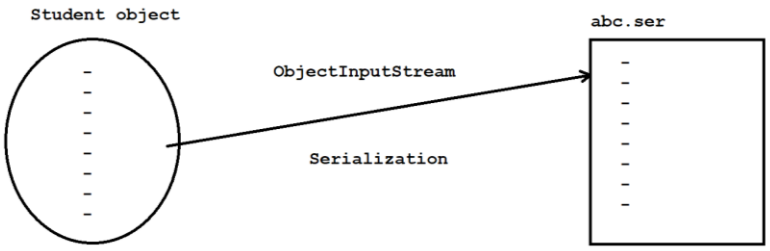

Diagram: jdbc1.1

In general, Serialization means converting object state to file state.

Deserialization
The process of taking the data from a file and represent an object is called deserialization.

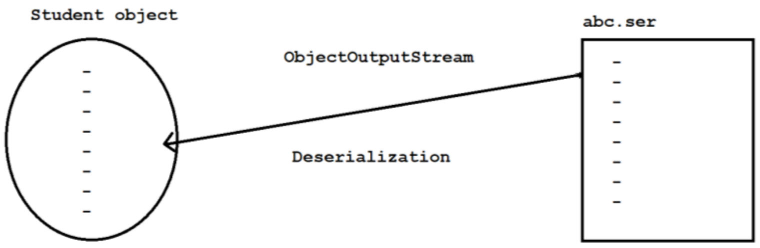

Diagram: jdbc1.2

Limitations with Files as persistence store
> It will store limited amount of data.
> There is no security.
> Fetching the data with multiple conditions is not possible.
> It does not show an application with relationship.
> It does not allow us to apply constraints (primary key, unique key, not null).
> Updating and Deletion of data can't be done directly.
> Merging and comparison of data can't be done easily.

Advantages of Database software as persistence store
> We can store unlimited amount of data.
> There is a security.
> It supports common query language.
> Fetching the data with multiple conditions is possible.
> It shows an application with relationships.
> It allows us to apply constraints.
> Deletion and Updating of data can be done directly.
> Merging and comparison of data can be done easily.

Every JDBC application is a two-tier application where java with JDBC code acts like a frontend/tier1/layer1 and database software acts like a backend/tier2/layer2.

Enduser is a non-technical person. He can't prepare and execute SQL query in database software. So he depends upon frontend developers having the capability to do that work for him.

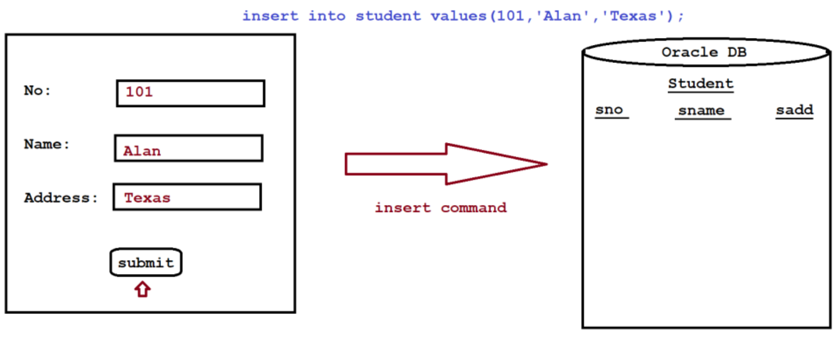

Diagram: jdbc1.3

JDBC Driver
It acts like a bridge between java application and database software.
It is used to convert Java calls to database calls and vice versa.
Here calls means instructions.

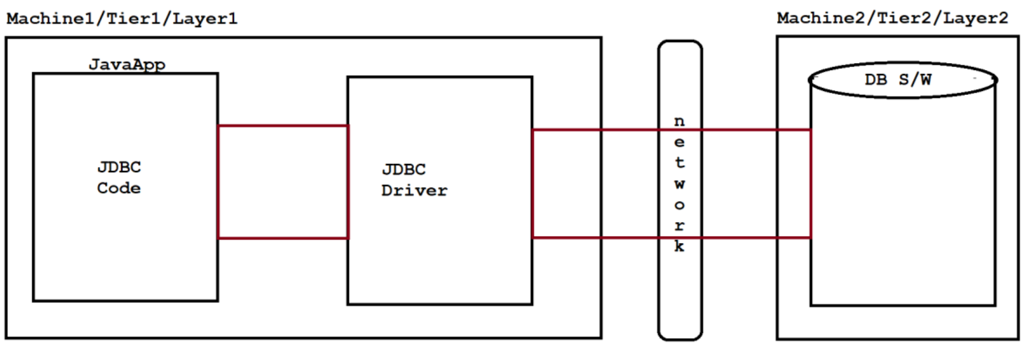

Diagram: jdbc1.4

ODBC Driver

VBScript, Perl, D2k and etc uses ODBC driver to interact with database software.

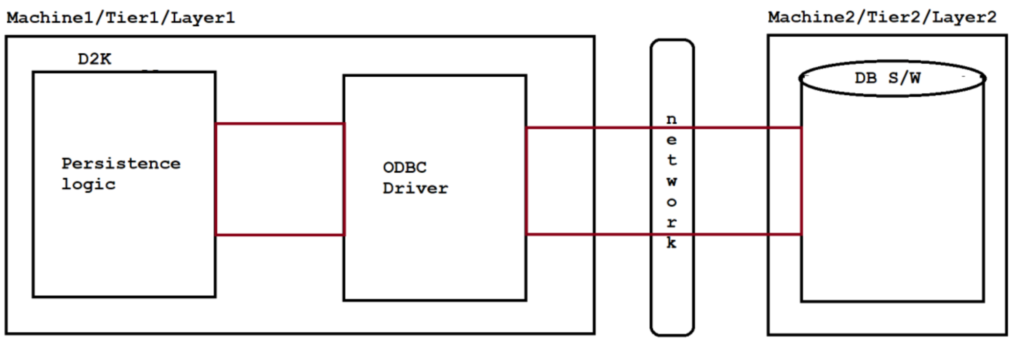

Diagram: jdbc1.5

ODBC driver is developed by using c language by taking the support of pointers. But Java does not support pointers. To overcome this limitation Sun Micro System introduced JDBC driver exclusively.

We will get JDBC software's from three parties.

1) Sun Micro System (creator of JDBC driver)
2) Database vendor
3) Third party vendor

We will get ODBC software's from three parties.

1) Xopen company (creator of ODBC driver)
2) Database vendor
3) Third party vendor

Q) What is JDBC?

It is a open technology given by Sun Micro System having set of rules and guidelines to develop JDBC drivers for multiple database software's.

Q) What is ODBC?

ODBC is a open technology given by Xopen company having set of rules and guidelines to develop ODBC drivers for multiple database software's.

Oracle

Version	:	10g or 11g
Vendor	:	Oracle corporation
Opensource	:	open source
Port No	:	1521
Username	:	system (default)
Password	:	admin
Website	:	https://www.oracle.com/in/database/

Download link :
https://drive.google.com/file/d/0B9rC21sL6v0td1NDZXpkUy1oMm8/view?usp=drive_link&resourcekey=0-aKooR3NmAh_eLo_qGw_inA

To use any JDBC driver we need to register with DriverManager service.

Every JDBC application contains one built-in service called DriverManager service.

Class.forName()
It always recommanded to use Class.forName() method to register JDBC driver with DriverManager service.
It is used to load the driver class but it won't create an object.

ex:
```
    Class.forName("driver-class-name");
```

Connection object
A Connection is an interface which is present in java.sql package.
It is an object of underlying supplied java class which implements java.sql.Connection interface.
If we want to interact with database we need to establish the connection with database software.
Once the work with database is completed, we need to close the Connection object.

ex:
```
    Connection con;
```

DriverManager.getConnection()
A DriverManager is a class which is present in java.sql package.
A getConnection() static method is used to interact with database software and returns JDBC Connection object representing connectivity between java application and database software.

ex:
```
    Connection con=DriverManager.getConnection("driver-url",username,pwd);
```

Statement object
A Statement is an interface which is present in java.sql package.
It acts like a vehicle between java application and database software.
It is used to sends and executes SQL query in database software.
We can create Statement object as follow.

ex:

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

ResultSet object
A ResultSet is an interface which is present in java.sql package.
Every ResultSet contains two positions.

1) BFR (Before First Record/Row)
2) ALR (After Last Record/Row)

Bydefault record pointer points to BFR position.
Every record ResultSet having 1 as base index and every record ResultSet having 1 as column index.

rs.next()
It will move record pointer to next position from current position.
If next position is a record then it will return true.
If next position is ALR then it will return false.
To read the values from record ResultSet we need to use getXxx() method with index number or column name.
Here getXxx() method means getInt(),getFloat(),getDouble() and etc.

```
rs.getInt(1);
rs.getString(2);
rs.getString(3);

or

rs.getInt("sno");
rs.getString("sname");
rs.getString("sadd");
```

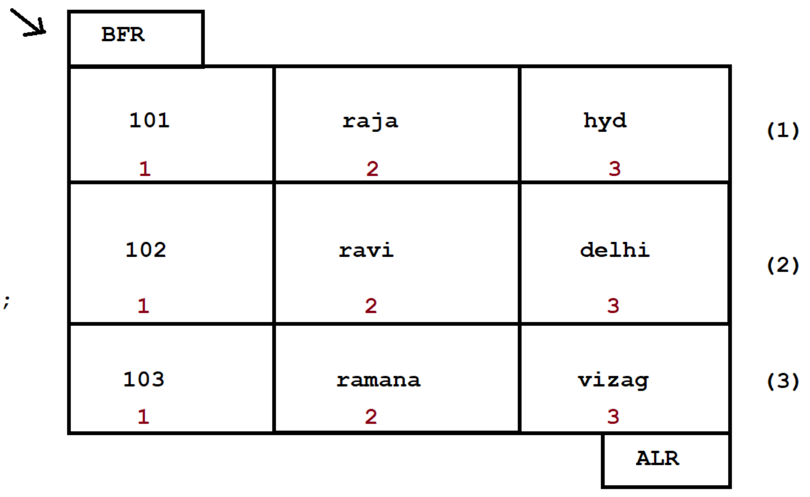

Diagram: jdbc2.1

student table

```
drop table student;

create table student(sno number(3),sname varchar2(10),sadd varchar2(12));

insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
insert into student values(103,'ramana','vizag');

commit;

select * from student;
```

Q)Difference between Oracle and MongoDB?

Oracle	**MongoDB**
It is a RDBMS database	MongoDB is a document base database.
table	Collection
row	document
column	field

Types of Queries in JDBC

According JDBC point of view we have two types of queries.

1)Select Query
2)Non-Select Query

1)Select Query

A select query will give bunch of records from database.

ex:

```
select * from student;
```

A JDBC Statement object gave executeQuery() method to execute select query.

ex:

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

2)Non-Select Query

A non-select query will return numeric value represent number of records effected in a database table.

ex:

```
delete from student;
```

A JDBC Statement object gave executeUpdate() method to execute non-select query.

ex:

```
int result=st.executeQuery("delete from student");
```

Steps to develop JDBC application

We have six steps to develop JDBC application.

1) Register JDBC driver with DriverManager service.
2) Establish the connection with database software.
3) Create Statement object.
4) Sends and Executes SQL query in database software.
5) Gather the result from database software to process the result.
6) Close all JDBC connection objects.

Q) How many drivers are there in JDBC?

We have four drivers in JDBC.

Type1 JDBC driver / JDBC-ODBC bridge driver
Type2 JDBC driver / Native API
Type3 JDBC driver / Net Protocol
Type4 JDBC driver / Native Protocol

Type4 JDBC driver / Native Protocol (Database properties)

```
Driver          : oracle.jdbc.driver.OracleDriver
                  ------------------       ------------
                            |                    |
                        pkg name       driver class name

URL             : jdbc:oracle:thin:@localhost:1521:XE
                  -----------------    |          |    |
                        |              |          |    |
                   sub protocol  hostname  portno  logical db name

Username        : system

Password        : admin
```

Eclipse

IDE	:	JEE IDE
Environment	:	Java Environment
Flavours	:	Kepler,Indigo,Luna,Mars and etc.
Vendor	:	Eclipse Foundation
Website	:	www.eclipse.org
Opensource	:	Open source
Format	:	Zip format
Download	:	

https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-ide-java-ee-developers

Note:
Extract the Eclipse software.
ex:
  right click to eclipse software --> exteract file -->
  select 'E' drive --> ok.

Steps to develop first JDBC application using Eclipse IDE
step1:
  Launch eclipse IDE by chossing workspace location.
step2:
  Create a java project i.e IH-JAVA-024.
  ex:
    File --> new --> project --> Java project --> Next -->
    Name : IH-JAVA-024 --> Next --> finish.
step3:
  Add "ojdbc14.jar" file in project build path.
  ex:
    Right click to IH-JAVA-024 project --> build path -->
    configuration build path --> libraries ---> Add external jars
    ---> select ojdbc14.jar file --> open --> ok.
step4:
  Create a "com.ihub.www" package inside "src" folder.
  ex:
    Right click to src folder --> new --> package -->
    Name : com.ihub.www --> finish.
step5:
  Create a JDBC application inside "com.ihub.www" package.
  ex:
    Right click to com.ihub.www package --> new -->
    class --> Name : SelectApp --> finish.

SelectApp.java
```
package com.ihub.www;

//ctrl+shift+o
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class SelectApp
{
	public static void main(String[] args)throws Exception
	{
			Class.forName("oracle.jdbc.driver.OracleDriver");

			Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

			Statement st=con.createStatement();

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

			while(rs.next())
			{
				System.out.println(rs.getInt(1)+"				"+rs.getString(2)+"
"+rs.getString(3));
			}

			rs.close();
			st.close();
			con.close();
		}
}
```

step6:
	Run the JDBC Application.
	ex:
		Right click to SelectApp.java --> run as --> Java Application.

Q)Write a JDBC application to select student name and student address based on student number?

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;

public class SelectApp2
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the student no :");
		int no=sc.nextInt();

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

		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		Statement st=con.createStatement();

		String qry="select sname,sadd from student where sno="+no;

		ResultSet rs=st.executeQuery(qry);

		int cnt=0;
		while(rs.next())
		{
			System.out.println(rs.getString(1)+" "+rs.getString(2));
			cnt=1;
		}

		if(cnt==0)
			System.out.println("No Rows Selected");

		rs.close();
		st.close();
		con.close();
	}
}
```

Non-Select Queries
Q)Write a JDBC application to insert a record into student table?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;
public class InsertApp {
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the student no :");
		int no=sc.nextInt();

		System.out.println("Enter the student name :");
		String name=sc.next();

		System.out.println("Enter the student address :");
		String add=sc.next();

		//convert inputs according to SQL query.
		name="'"+name+"'";
		add="'"+add+"'";

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

		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		Statement st=con.createStatement();

		String qry="insert into student values("+no+","+name+","+add+")";

		int result=st.executeUpdate(qry);

		if(result==0)
			System.out.println("No Record Inserted");
		else
			System.out.println(result+" Record Inserted");

		st.close();
		con.close();

	}
}
```
Q)Write a JDBC application to update student name based on student number?

```
package com.ihub.www;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;

public class UpdateApp {

        public static void main(String[] args)throws Exception
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("Enter the student name :");
                String name=sc.next();

                System.out.println("Enter the student no :");
                int no=sc.nextInt();

                //converting inputs according to SQL query
                name="'"+name+"'";

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

                Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

                Statement st=con.createStatement();

                String qry="update student set sname="+name+" where sno="+no;

                int result=st.executeUpdate(qry);

                if(result==0)
                        System.out.println("No Record updated");
                else
                        System.out.println(result+" Record updated");

                st.close();
                con.close();
        }
}
```

Q)Write a jdbc application to delete a student record based on student no?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;

public class DeleteApp {

	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the student no :");
		int no=sc.nextInt();

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

		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		Statement st=con.createStatement();

		String qry="delete from student where sno="+no;

		int result=st.executeUpdate(qry);

		if(result==0)
			System.out.println("No Record deleted");
		else
			System.out.println(result+" Record deleted");

		st.close();
		con.close();
	}
}
```

Q)Types of Statement objects in JDBC?

We have three Statement objects in JDBC.

1) Simple Statement

It is an object of underlying supplied java class which implements java.sql.Statement interface.

2) PreparedStatement

It is an object of underlying supplied java class which implements java.sql.PreparedStatement interface.

3) CallableStatement

It is an object of underlying supplied java class which implements java.sql.CallableStatement interface.

SQL Injection problem

Along with input values if we pass special SQL instructions which change behaviour of a query and behaviour of an application is called SQL injection problem.

Here special SQL instruction means comment in SQL i.e (--).

While dealing with simple Statement object there is a chance of raising SQL injection problem.

ex:

```
Username :  raja'--
password :  hyd

Valid Credentials
```

Userlist table

```
drop table userlist;
create table userlist(uname varchar2(10),pwd varchar2(10));
insert into userlist values('raja','rani');
insert into userlist values('king','kingdom');
commit;
```

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;
```

```
public class SQLInjProbApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the username :");
		String name=sc.next();

		System.out.println("Enter the password :");
		String pass=sc.next();

		//converting inputs according to SQL query
		name="'"+name+"'";
		pass="'"+pass+"'";

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		Statement st=con.createStatement();

		String qry="select count(*) from userlist where uname="+name+" and
pwd="+pass;

		ResultSet rs=st.executeQuery(qry);

		int result=0;
		while(rs.next())
		{
			result=rs.getInt(1);
		}

		if(result==0)
			System.out.println("Invalid Credentials");
		else
			System.out.println("Valid Credentials");

		rs.close();
		st.close();
		con.close();
	}
}
```

Type1 JDB Driver Architecture/JDBC-ODBC Bridge Driver (Partly java driver)
Type1 JDBC driver is not designed to interact with database software directly.
It is designed to take the support of ODBC drivers and Vendor DB libraries to locate and interact with database software.

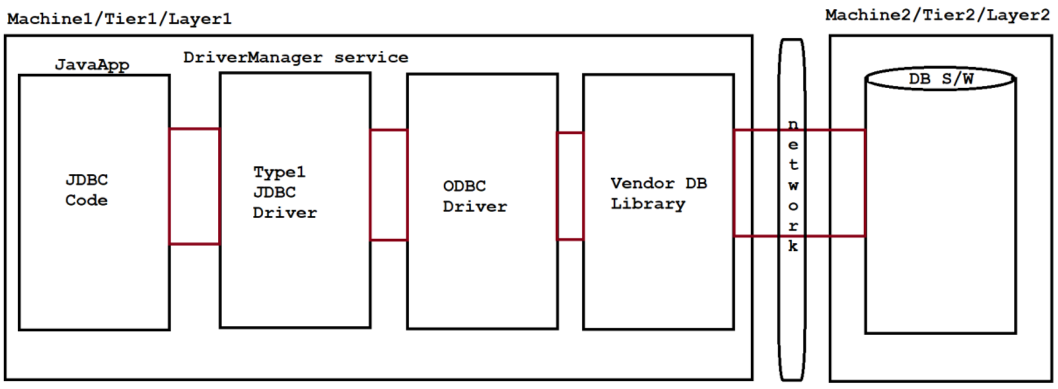

Diagram: jdbc4.1

Advantages:
> It is a built-in driver of JDK.
> Using Type1 JDBC driver we can interact with any database software.

Disadvantages:
> This driver performance is low.It is not suitable for medium and large scale applications.Hence it is not a industry standard driver.
> To work with type1 jdbc driver we need to arrange ODBC driver and vendor db library.
> Since ODBC driver and vendor db library present at client side so it is not suitable for untrusted applets to database communication.

Type2 JDBC Driver Architecture / Native API (partly java driver)
Type2 JDBC driver is not designed to interact with database software directly.
It is designed to take the support of vendor db library to locate and interact with database softwares.

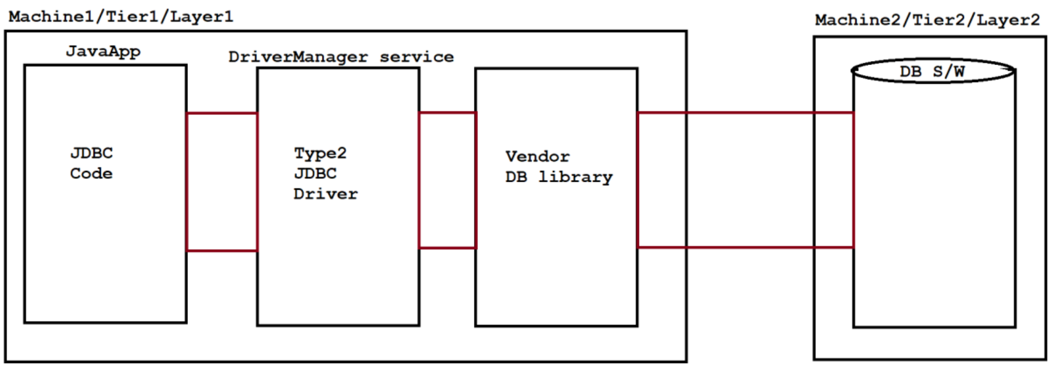

Diagram: jdbc4.2

Advantages:
> This driver will give better performance when compare to Type1 JDBC driver.
> It will not take the support of ODBC driver.

Disadvantages:

> This driver performance is quit slow.It is not suitable for medium and large scale projects.Hence it is not a industry standard driver.
> To work with Type2 JDBC driver we need to arrange vendor db library seperately.
> Since vendor db library present at client side so it is not suitable for untrusted applets to database communication.
> For every database software we need to arrange Type2 jdbc driver seperately.

Type4 JDBC driver / Native Protocol (Java driver) / thin driver

Type4 JDBC driver is not designed to take the support of ODBC driver and vendor db library.
It is designed to interact with database software directly.

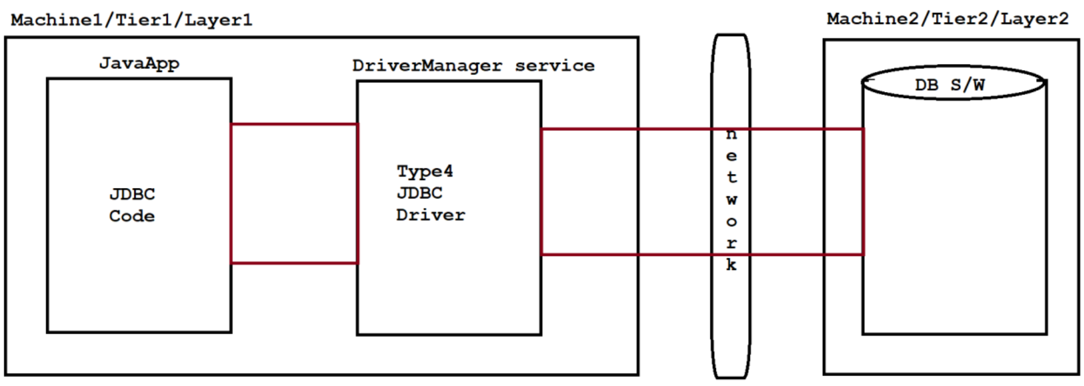

Diagram: jdbc4.3

Advantages:

> This driver will give better performance when compare to Type 1 & 2 driver.
> It is developed in java so it will give platform independency.
> It will not take the support of odbc driver and vendor db library.
> It is suitable for medium and large scale project.Hence it is a industry standard driver.
> Since odbc driver and vendor db library not present at client side so it is suitable for untrusted applets to database communication.

Disadvantages:

> It is a built-in driver of JDK.
> For every database we need to arrange type4 jdbc driver seperately.

JDBC Connection pool

It is a factory containing set of readily avaiable JDBC Connection objects before actual being used.

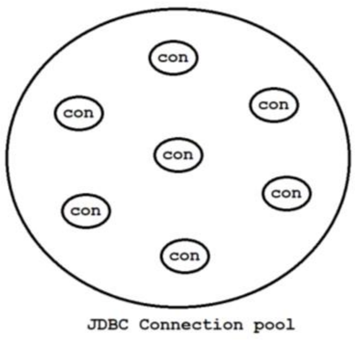

Diagram: jdbc4.4

JDBC Connection pool represent connectivity with same database software.

Advantages:

> It will give resuable JDBC Connection objects.

> With minimum number of Connection objects we can interact with multiple clients.

> A user is not responsible to create,manage and destroy JDBC connection objects. A JDBC Connection pool is responsible to create , manage and destroy jdbc Connection object in JDBC Connection pool.

Type3 JDBC Driver Architecture / Net Protocol

Web server, Proxy server or IDE's server contains JDBC Connection pool representing reusable JDBC Connection objects.

Type3 JDBC driver is not designed to interact with database software directly.

Type3 JDBC driver is designed to interact with web server or proxy server to get one reusable JDBC Connection object from JDBC Connection pool.

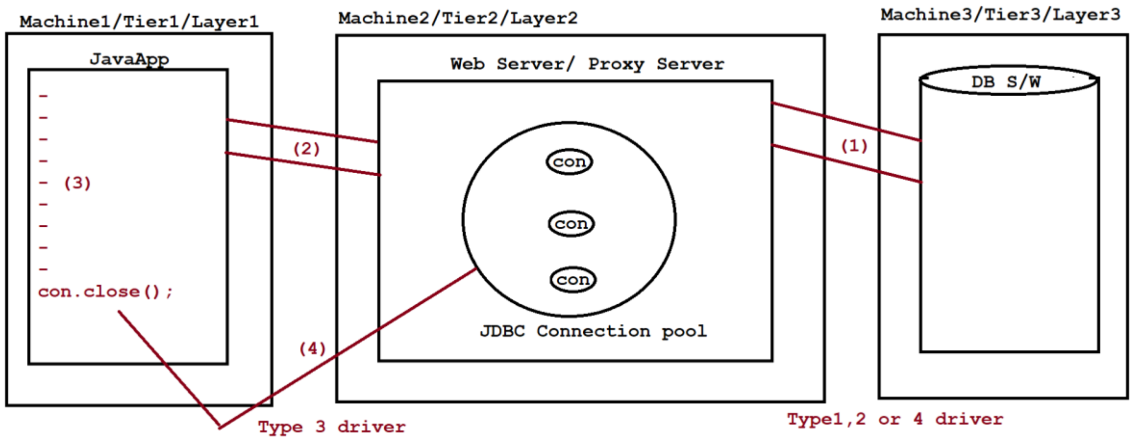

Diagram: jdbc5.1

With respect to the diagram:

1) Webserver or proxy server interacts with database software and gets JDBC Connection objects in JDBC Connection pool.

2) Java application interacts with web server or proxy server and gets one reusable JDBC Connection object from JDBC Connection pool.

3) Our application uses JDBC Connection object to create other connection objects.

4) Once if we call con.close() then JDBC Connection object goes back to JDBC Connection pool.

Q) How many JDBC Connection objects are there in JDBC?

We have two JDBC Connection objects.
1) Direct JDBC Connection object
A JDBC Connection object which is created by the user is called direct JDBC Connection object.
ex:
```
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection con=DriverManager.getConnection
    ("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
```

2) Pooled JDBC Connection object
A JDBC Connection object which is gathered from JDBC Connection pool is called
pooled JDBC Connection object.

Limitations with simple Statement object
> It is not suitable to perform same query for multiple times with same values or different values.
> We can't use String values directly to query parameter without any conversion.
> Framing query with variables is quit complex.
> It raises SQL injection problem.
> It does not allow us to insert date values in a database table column.
> It does not allow us to insert LOB values in a database table column.

To overcome this limitations we need to use PreparedStatement object.

Pre-compiled SQL Query
Our query goes to database software without inputs and becomes parsed query
either it is executed or not is called pre-compiled SQL query.
PreparedStatment object deals with pre-compiled SQL query.

Working with PreparedStatement object
step1:
    Create a query with placeholders or parameters.
    ex:
```
        String qry="insert into student values(?,?,?)";
```
step2:
    Convert SQL query to precompiled SQL query.
    ex:
```
        PreparedStatement ps=con.prepareStatement(qry);
```
step3:
    Set the values to query parameters.
    ex:
```
        ps.setInt(1,no);
```

```
                ps.setString(2,name);
                ps.setString(3,add);
step4:
        Execute pre-compiled SQL Query.
        ex:
                ps.executeUpdate();
step5:
        Close PreparedStatement object.
        ex:
                ps.close();
```

Q)Write a jdbc application to insert a record into student table using PreparedStatement object?

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;
public class PSInsertApp
{
        public static void main(String[] args)throws Exception
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student no :");
                int no=sc.nextInt();
                System.out.println("Enter the student name :");
                String name=sc.next();
                System.out.println("Enter the student address :");
                String add=sc.next();

                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
                String qry="insert into student values(?,?,?)";
                PreparedStatement ps=con.prepareStatement(qry);
                //set the values to place holders
                ps.setInt(1,no);
                ps.setString(2,name);
                ps.setString(3,add);
                //execute
                int result=ps.executeUpdate();
                if(result==0)
                    System.out.println("Record Not Inserted");
                else
                    System.out.println("Record Inserted");

                ps.close();
                con.close();
        }
}
```
Q)Write a jdbc application to update student name based on student number?

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;
public class PSUpdateApp
{
        public static void main(String[] args)throws Exception
        {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the student no :");
            int no=sc.nextInt();
            System.out.println("Enter the student name :");
            String name=sc.next();

            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

            String qry="update student set sname=? where sno=?";
            PreparedStatement ps=con.prepareStatement(qry);

            //set the values
            ps.setString(1,name);
            ps.setInt(2,no);

            //execute
            int result=ps.executeUpdate();

            if(result==0)
                System.out.println("No Record Updated");
            else
                System.out.println("Record Updated");
```

```
		ps.close();
		con.close();
	}
}
```
Q)Write a JDBC Application to delete a student record based on student number?

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PSDeleteApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the student no :");
		int no=sc.nextInt();

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		String qry="delete from student where sno=?";

		PreparedStatement ps=con.prepareStatement(qry);

		//set the values
		ps.setInt(1,no);

		//execute
		int result=ps.executeUpdate();

		if(result==0)
			System.out.println("No Record Deleted");
		else
			System.out.println("Record Deleted");

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

```
          }
}
```

Solution for SQL Injection problem

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

public class SolForSQLInjProb
{
          public static void main(String[] args)throws Exception
          {
                    Scanner sc=new Scanner(System.in);
                    System.out.println("Enter the username :");
                    String name=sc.next();
                    System.out.println("Enter the password :");
                    String pass=sc.next();

                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

                    String qry="select count(*) from userlist where uname=? and pwd=?";

                    PreparedStatement ps=con.prepareStatement(qry);

                    //set the values
                    ps.setString(1,name);
                    ps.setString(2,pass);

                    //execute
                    ResultSet rs=ps.executeQuery();

                    int result=0;
                    while(rs.next())
                    {
                              result=rs.getInt(1);
                    }
                    if(result==0)
```

```
				System.out.println("Invalid Credentials ");
			else
				System.out.println("Valid Credentials ");

			rs.close();
			ps.close();
			con.close();
		}
}
```

DatabaseMetaData

DatabaseMetaData is an interface which is present in java.sql package.
DatabaseMetaData provides metadata of a database.
DatabaseMetaData gives information about database product name, database product version, database driver name, database driver version, database username and etc.
We can create DatabaseMetaData object as follow.
ex:

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

ex:

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
public class DBMDApp {

		public static void main(String[] args)throws Exception
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");
			Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

			DatabaseMetaData dbmd=con.getMetaData();

			System.out.println(dbmd.getDatabaseProductName());
			System.out.println(dbmd.getDatabaseProductVersion());
			System.out.println(dbmd.getDriverName());
			System.out.println(dbmd.getDriverVersion());
			System.out.println(dbmd.getUserName());
			con.close();
		}

}
```

ResultSetMetaData
ResultSetMetaData is an interface which is present in java.sql package.
ResultSetMetaData provides metadata of a table.
ResultSetMetaData gives information about number of columns , name of a columns, type of columns, size of a columns and etc.
We can create ResultSetMetaData object as follow.
ex:
```
	ResultSetMetaData rsmd=rs.getMetaData();
```

ex:
```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class RSMDApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
		Statement st=con.createStatement();
		String qry="select * from student";
		ResultSet rs=st.executeQuery(qry);

		ResultSetMetaData rsmd=rs.getMetaData();

		System.out.println(rsmd.getColumnCount());
		System.out.println(rsmd.getColumnName(1));
		System.out.println(rsmd.getColumnTypeName(2));
		System.out.println(rsmd.getColumnDisplaySize(2));

		rs.close();
		st.close();
		con.close();

	}
}
```

Working with Date values

While dealing with DOB,DOA,DOR,DOD and etc we need to insert and retrieve date values.

It is never recommanded to store date values in the form of String because we can't compare two dates.

Every database software supports different date patterns.

ex:

Oracle - dd-MMM-yy

MySQL - yyyy-MM-dd

A java.util.Date class object is not suitable to perform database operation.

A java.sql.Date class object is suitable to perform database operation.

Using simple Statement object we can place date values to query parameters.

To overcome this limitation we need to use PreparedStatement object.

Once JDBC driver will get the date value then it will insert in the pattern which is supported by underlying database software.

Standard procedure to insert date

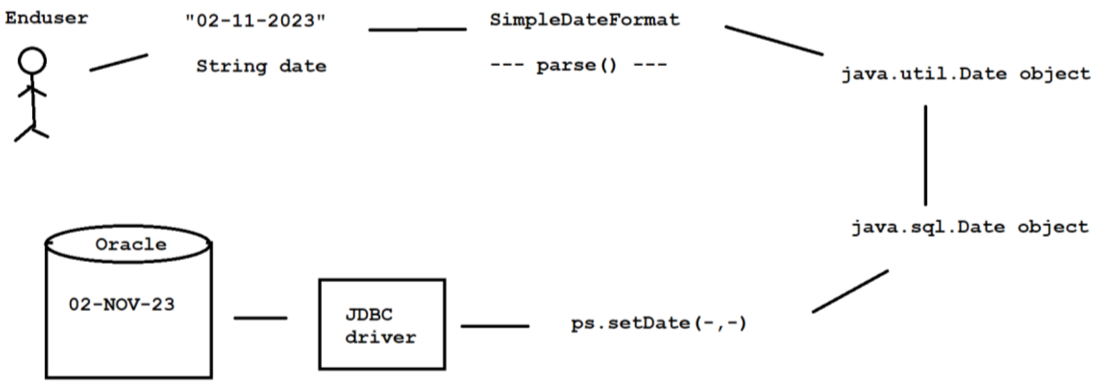

Diagram: jdbc6.1

With the respect to the diagram:

1) Enduser will pass date value in the form of String.

2) A parse() method of java.text.SimpleDateFormat class converts String date to java.util.Date class object.

3) Our application converts java.util.Date class object to java.sql.Date classobject.

4) A ps.setDate(-,-) method is used to set date value to query parameter.

5) Once JDBC driver gets date value then it will insert in the pattern which is supported by underlying database software.

<u>emp1 table</u>

drop table emp1;

create table emp1(eid number(3),ename varchar2(10),edoj date);

DateInsertApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.text.SimpleDateFormat;
import java.util.Scanner;

public class DateInsertApp {

	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the employee id ");
		int id=sc.nextInt();

		System.out.println("Enter the employee name :");
		String name=sc.next();

		System.out.println("Enter the employee DOJ(dd-MM-yyyy) :");
		String sdoj=sc.next();

		//converting string date to java.util.Date class object
		SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
		java.util.Date udoj=sdf.parse(sdoj);

		//converting util date object to sql date object
		long ms=udoj.getTime();
		java.sql.Date sqldoj=new java.sql.Date(ms);

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		String qry="insert into emp1 values(?,?,?)";

		PreparedStatement ps=con.prepareStatement(qry);

		//set the values
		ps.setInt(1,id);
		ps.setString(2,name);
		ps.setDate(3,sqldoj);

		//execute
		int result=ps.executeUpdate();

		if(result==0)
			System.out.println("No Record inserted");
		else
			System.out.println("Record inserted");

		ps.close();
		con.close();

	}

}
```

DateRetrieveApp.java

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;

public class DateRetrieveApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
		Statement st=con.createStatement();
		String qry="select * from emp1";
		ResultSet rs=st.executeQuery(qry);
		while(rs.next())
		{
			int id=rs.getInt(1);
			String name=rs.getString(2);
			java.sql.Date sqldoj=rs.getDate(3);

			//converting sql date to util date
```

```
                        java.util.Date udoj=(java.util.Date)sqldoj;

                        SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
                        String sdoj=sdf.format(udoj);

                        System.out.println(id+" "+name+" "+sdoj);
                }

                rs.close();
                st.close();
                con.close();


        }
}
```

Working with LOB values
Files are known as LOB's.
We have two types of LOB's.

1) BLOB (Binary Large Object)
ex:
images,audio,video,avi file and etc.
2)CLOB (Character Large Object)
ex:
text,doc file ,advanced text file and etc.

While dealing with matrimonial applications, job portal applications, profile management applications and etc. we need to insert and retrieve LOB values.
Using simple Statement object we can set LOB values directly to query parameters.We need to take the support of PreparedStatement object.
We can set LOB values to query parameter by using following methods.
ex:

```
        ps.setBinaryStream(-,-,-) / ps.setBLOB(-,-,-)
        ps.setCharacterStream(-,-,-) / ps.setCLOB(-,-,-)
```

emp2 table
==========

```
drop table emp2;

create table emp2(eid number(3),ename varchar2(10), ephoto BLOB);
```

PhotoInsertApp.java

```
package com.ihub.www;

import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PhotoInsertApp {

	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the employee id :");
		int id=sc.nextInt();

		System.out.println("Enter the employee name :");
		String name=sc.next();

		//locate the file
		File f=new File("src/com/ihub/www/rock.jpeg");
		FileInputStream fis=new FileInputStream(f);

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

		String qry="insert into emp2 values(?,?,?)";

		PreparedStatement ps=con.prepareStatement(qry);

		//set the values
		ps.setInt(1,id);
		ps.setString(2,name);
		ps.setBinaryStream(3,fis,(int)f.length());

		//execute
		int result=ps.executeUpdate();
		if(result==0)
			System.out.println("No Record inserted");
		else
```

```
                              System.out.println("Record inserted");

                    ps.close();
                    con.close();

          }

}
```

PhotoRetrieveApp.java

```
package com.ihub.www;

import java.io.FileOutputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class PhotoRetrieveApp
{
          public static void main(String[] args)throws Exception
          {
                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
                    Statement st=con.createStatement();

                    String qry="select * from emp2";

                    ResultSet rs=st.executeQuery(qry);

                    while(rs.next())
                    {
                              InputStream is=rs.getBinaryStream(3);

                              FileOutputStream    fos=new    FileOutputStream("E:\\IHUB-Training-
Batches\\IH-JAVA-024\\praveen.png");

                              int byteReads=0;

                              byte[] buff=new byte[255];
```

```
				while((byteReads=is.read(buff))!=-1)
				{
					fos.write(buff, 0, byteReads);
				}

				fos.close();
			}

			System.out.println("Please check the location");
			rs.close();
			st.close();
			con.close();

		}
}
```

JDBC Flexible Application
In JDBC , Connection object consider as heavy weight object.
It is never recommanded to create JDBC Connection object in every JDBC application.
We need to a create a class which returns JDBC Connection object.

DBConnection.java

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
public class DBConnection
{
	static Connection con=null;
	public static Connection getConnection()
	{
		try
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");
	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system",
"admin");
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}

		return con;
	}
}
```

FlexibleApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class FlexibleApp
{
	public static void main(String[] args)throws Exception
	{
		Connection con=DBConnection.getConnection();

		Statement st=con.createStatement();

		String qry="select * from student";

		ResultSet rs=st.executeQuery(qry);

		while(rs.next())
		{
			System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
		}

		rs.close();
		st.close();
		con.close();
	}
}
```

Working with Properties file

In regular intervals, Our DBA will change username and password for security reasons.
It is never recommanded to pass database properties directly to the application.
It is always recommanded to read database properties from properties file.
A properties file contains key and value pair.

dbdetails.properties

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:XE
username=system
password=admin
```

PropertiesApp.java

```
package com.ihub.www;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;
public class PropertiesApp
{
	public static void main(String[] args)throws Exception
	{
		//locate properties file
		FileInputStream                                        fis=new
FileInputStream("src/com/ihub/www/dbdetails.properties");

		//create Properties class object
		Properties p=new Properties();

		//reading the data from file to class
		p.load(fis);

		//reading the data from class
		String s1=p.getProperty("driver");
		String s2=p.getProperty("url");
		String s3=p.getProperty("username");
		String s4=p.getProperty("password");

		Class.forName(s1);
		Connection con=DriverManager.getConnection(s2,s3,s4);
		Statement st=con.createStatement();
		String qry="select * from student";
		ResultSet rs=st.executeQuery(qry);
		while(rs.next())
		{
			System.out.println(rs.getInt(1)+"              "+rs.getString(2)+"
"+rs.getString(3));
		}
		rs.close();
		st.close();
		con.close();
	}
}
```

Thin-Client/Fat-Server Application

Every JDBC application consider as Thin-Client/Fat-Server application.
To develop thin-client/fat-server application , we need to save our business logic and presistence logic in database software in the form of PL/SQL procedures and Functions.
To deal with PL/SQL procedures and functions we need to use CallableStatement object.

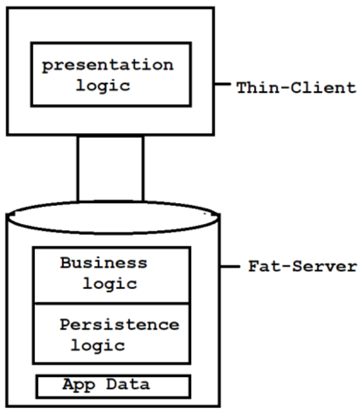

Diagram: jdbc7.1

PL/SQL procedure

```
create or replace procedure first_proc(A IN number,B IN number,C OUT number)
IS
BEGIN
C:=A+B;
END;
/

package com.ihub.www;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;

public class CallableStmtApp
{
        public static void main(String[] args)throws Exception
        {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

                CallableStatement cst=con.prepareCall("{CALL first_proc(?,?,?)}");

                //register OUT parameter
                cst.registerOutParameter(3, Types.INTEGER);

                //set the values to IN parameter
                cst.setInt(1, 10);
                cst.setInt(2, 20);

                //execute
                cst.execute();

                //gather the result
                int result=cst.getInt(3);
                System.out.println("sum of two numbers is ="+result);

                cst.close();
                con.close();
        }
}
```

Types of ResultSet Objects
We have two types of ResultSet objects.
1)Non-Scrollable ResultSet object
2)Scrollable ResultSet object

<u>1) Non-Scrollable ResultSet object</u>
A ResultSet object which allows us to read the records sequentially, unidirectionally is called non-scrollable ResultSet object.
Bydefault every ResultSet object is a non-scrollable ResultSet object.
If JDBC Statement object is created without type,mode value then ResultSet object is called Non-scrollable ResultSet object.
ex:

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

<u>2) Scrollable ResultSet object</u>
A ResultSet object which allows us to read the records non-sequentially, bidirectionally, randomly is called scrollable ResultSet object.
If JDBC Statement object is created with type,mode value then ResultSet object is called Non-scrollable ResultSet object.
ex:

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

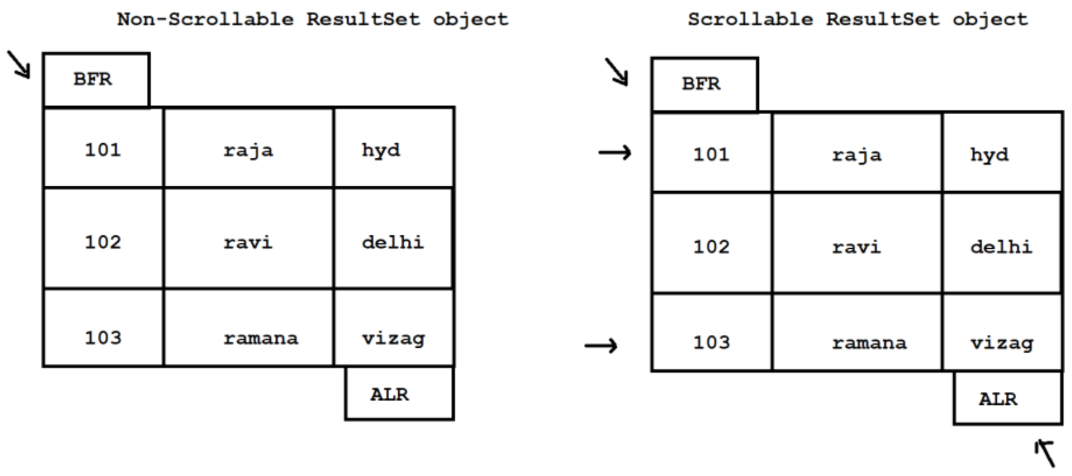

Diagram: jdbc8.1

We have two type values.
ex:

    ResultSet.TYPE_SCROLL_SENSITIVE
    ResultSet.TYPE_SCROLL.INSENSITIVE

We have two mode values.
ex:

    ResultSet.CONCUR_READ_ONLY
    ResultSet.CONCUR_UPDATABLE

Various methods present in Scrollable ResultSet object

rs.next()
It will move the record pointer to next position.
rs.getRow()
It will return position of record pointer.
rs.getXxx()
It will return the values from record ResultSet.
rs.close()
It is used to close the ResultSet object.
rs.previous()
It will move the record pointer to previous position.
rs.first()
It will set the record pointer to first record.

rs.isFirst()
It is used to check record pointer is in first position or not.
rs.last()
It will set the record pointer to last record.
rs.isLast()
It is used to check record pointer is in last position or not.

rs.beforeFirst()

It will set the record pointer to BFR position.

rs.afterLast()

It will set the record pointer to ALR position.

rs.relative(+/-)

It will move the record pointer to next position based on current position.

rs.absolute(+/-)

It will move the record pointer to next position based on BFR and ALR.

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class ScrollableResultSetApp {

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

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
		Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,

	ResultSet.CONCUR_READ_ONLY);

		String qry="select * from student";

		ResultSet rs=st.executeQuery(qry);

		//top to bottom
		while(rs.next())
		{
			System.out.println(rs.getRow()+"  "+rs.getInt(1)+"  "+rs.getString(2)+"
"+rs.getString(3));
		}

		rs.afterLast();
```

```
                    while(rs.previous())
                    {
                              System.out.println(rs.getRow()+"  "+rs.getInt(1)+"  "+rs.getString(2)+"
"+rs.getString(3));
                    }

                    rs.first();
                    System.out.println(rs.isFirst());
                    System.out.println(rs.getRow()+"       "+rs.getInt(1)+"       "+rs.getString(2)+"
"+rs.getString(3));

                    rs.last();
                    System.out.println(rs.isLast());//true
                    System.out.println(rs.getRow()+"       "+rs.getInt(1)+"       "+rs.getString(2)+"
"+rs.getString(3));

                    //rs.relative(-2);
                    rs.absolute(-2);
                    System.out.println(rs.getRow()+"       "+rs.getInt(1)+"       "+rs.getString(2)+"
"+rs.getString(3));

                    rs.close();
                    st.close();
                    con.close();

          }

}
```

Batch Processing
Batch processing is used to declare multiple queries in the application and makes a single call to the database.
Each query we need to add in a batch.
To add the query in a batch we need to use addBatch() method Statement object.
ex:
```
          st.addBatch("select * from student");
```

To execute the batch we need to use executeBatch() method of Statement object.
ex:
```
          int[] result=st.executeBatch();
```

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class BatchProcessing
{
	public static void main(String[] args)throws Exception
	{

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");
		Statement st=con.createStatement();

		String qry1="insert into student values(104,'ramulu','pune')";
		String qry2="delete from student where sno=103";
		String qry3="update student set sname='rani' where sno=101";

		//add the queries to batch
		st.addBatch(qry1);
		st.addBatch(qry2);
		st.addBatch(qry3);

		//execute the batch
		int[] result=st.executeBatch();

		//for each loop
		int sum=0;
		for(int i:result)
		{
			sum+=i;
		}
		System.out.println("No of records effected are ="+sum);

		st.close();
		con.close();
	}
}
```

Transaction Management

Transaction represent single unit of work.

JDBC Connection interface methods we can manage the transaction management.

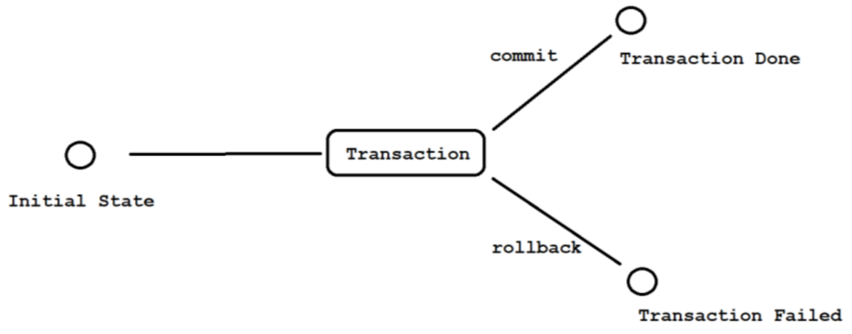

Diagram: jdbc9.1

<u>sbi bank</u>

```
drop table sbi;
create table sbi(accno number(6),accholder varchar2(10), accbal number(10));
insert into sbi values(111111,'sandeep',10000);
insert into sbi values(222222,'pradeep',12000);
commit;
```

<u>kotak bank</u>

```
drop table kotak;
create table kotak(accno number(6),accholder varchar2(10), accbal number(10));
insert into kotak values(100001,'thrishul',90000);
insert into kotak values(200002,'raju',80000);
commit;
```

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;
public class TXNManagementApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the source account no :");
		int sno=sc.nextInt();
		System.out.println("Enter the destination account no :");
		int dno=sc.nextInt();
		System.out.println("Enter the amount to transfer :");
		int amt=sc.nextInt();
		Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
                    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin
");

                    //set auto commit false
                    con.setAutoCommit(false);

                    Statement st=con.createStatement();

                    //create the queries
                    String qry1="update kotak set accbal=accbal-"+amt+" where accno="+sno;
                    String qry2="update sbi set accbal=accbal+"+amt+" where accno="+dno;

                    //add the queries to batch
                    st.addBatch(qry1);
                    st.addBatch(qry2);

                    //execute the batch
                    int[] result=st.executeBatch();

                    boolean flag=true;
                    for(int i:result)
                    {
                              if(i==0)
                              {
                                        flag=false;
                                        break;
                              }
                    }
                    if(flag==true)
                    {
                              System.out.println("Transaction Done Successfully");
                              con.commit();
                    }
                    else
                    {
                              System.out.println("Transaction Failed!!");
                              con.rollback();
                    }

                    st.close();
                    con.close();
          }
}
```

Standard procedure to develop JDBC application

```
package com.ihub.www;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class StandardApp
{
	public static void main(String[] args)
	{
		final String DRIVER="oracle.jdbc.driver.OracleDriver";
		final String URL="jdbc:oracle:thin:@localhost:1521:XE";
		final String USERNAME="system";
		final String PASSWORD="admin";


		Connection con=null;
		Statement st=null;
		ResultSet rs=null;
		String qry=null;
		try
		{
			Class.forName(DRIVER);

	con=DriverManager.getConnection(URL,USERNAME,PASSWORD);
			st=con.createStatement();
			qry="select * from student";
			rs=st.executeQuery(qry);
			while(rs.next())
			{
				System.out.println(rs.getRow()+"                    "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
			}

			rs.close();
			st.close();
			con.close();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

Steps to interact with MYSQL Database

step1:

Download and Install MY/SQL Database successully.

ex:

https://drive.google.com/file/d/1QQjWTJ9v8xz0nfuSGva1_QQwO6KDf9_c/view?usp=sharing

step2:

Connect with mysql by using password.

ex:

username : root(default)

password: root

step3:

create a SCHEMA in MYSQL.

ex:

```
create schema IH_JAVA_024
```

step4:

To check list of databases /schemas present in mysql db.

ex:

```
show databases;
```

step5:

Use IH_JAVA_024 scheme/database.

ex:

```
use   IH_JAVA_024;
```

step6:

create a student table and insert the records.

ex:

```
create table student(sno int(3),sname varchar(10),sadd varchar(10));
insert into student values(101,'raja','hyd');
insert into student values(102,'raju','delhi');
insert into student values(103,'ravi','pune');
commit;
```

step7:

create a JDBC Application to select student records.

MySQLApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class MySQLApp
{
	public static void main(String[] args)
	{
		final String DRIVER="com.mysql.jdbc.Driver";
final String URL="jdbc:mysql://localhost:3306/IH_JAVA_024?characterEncoding=utf8";
		final String USERNAME="root";
		final String PASSWORD="root";
		final String QUERY="select * from student";

		Connection con=null;
		Statement st=null;
		ResultSet rs=null;
		try
		{
			Class.forName(DRIVER);

	con=DriverManager.getConnection(URL,USERNAME,PASSWORD);
			st=con.createStatement();
			rs=st.executeQuery(QUERY);
			while(rs.next())
			{
				System.out.println(rs.getRow()+"			"+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
			}

			rs.close();
			st.close();
			con.close();
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

step8:

Add "mysql-connector.jar" file in project build path for mysql database.

right click to project --> built path --> configuration build path --> libraries
--> add external jars --> select mysql-connector.jar file --> open.

jar file download :
http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjavajar.htm
http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjar.htm

Note:
ojdbc14.jar - - for oracle

mysql-connector.jar --> for mysql
or
mysql-connector-java.jar --> for mysql

step9:

Run the jdbc application.

SERVLETS

Web Application

A web application is a collection of web resource programs having the capability to generate web pages.

We have two types of web pages.

1) static web page / passive web page

If a content not change for a web page is called static web page.

ex:

- facebook login page
- home page
- contactus page
- services page
- and etc

2) Dynamic web page / active web page

If a content change for a web page is called dynamic web page.

ex:

- live cricket score page
- stock market share value page
- gmail inbox page
- and etc

We have two types of web resource programs.

1) Static web resource program

It is used to generate static web pages.

ex:

- html program
- css program
- bootstrap program
- reactjs program
- angularjs program
- and etc.

2) Dynamic web resource program

It is used to generate dynamic web pages.

ex:

- servlet program
- jsp program
- and etc.

Based onthe position and execution these web resource programs are dividedi into two types.

1) Client side web resource program

A web resource program which executes at client side(browser window) is called client side web resource program.

All static web resource programs are called client side web resource program.

2) Server side web resource program

A web resource program which executes at server side is called server side web resource program.

All dynamic web resource programs are called server side web resource program.

Web application and web resource program execution

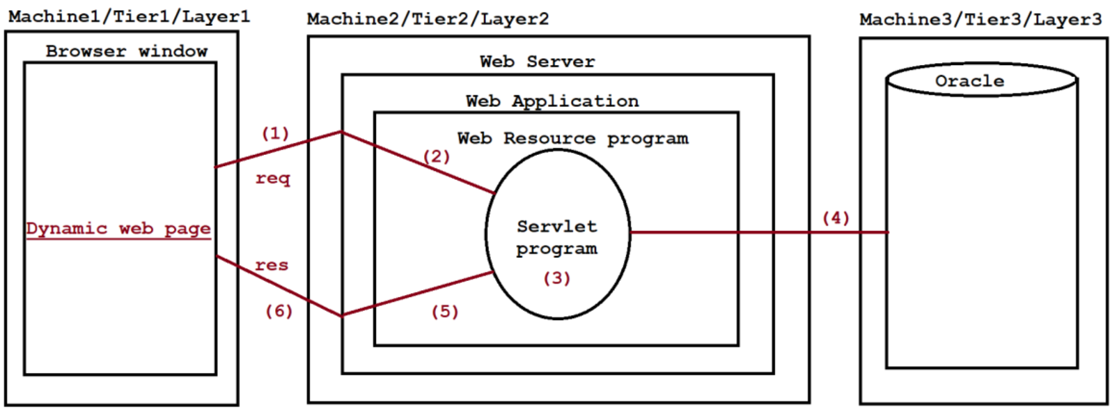

Diagram: servlet1.1

Java application will execute manually.

Web application and web resource program will execute at the time when they have requested. So there is no way to execute them manually.

With respect to the diagram:

1) Enduser will give the request to web resource program.

2) Web server will trap that request and passes that request to appropriate web resource program.

3) Web resource program will execute the logic to process the request.

4) Web resource program will communicate with database software if neccessary.

5) Web resource program sends the output to web server.

6) Web server will give output to browser window as dynamic response.

Note:

The process of keeping the web application in a server is called deployment and reverse is called undeployment.

Web Server

A web server is a piece of software which is used to automate whole process of web application and web resource program execution.

ex:

Tomcat , Resin and etc.

Responsibilities of web server
> It takes continues request from client.
> It passes the request to appropriate web resource program.
> It provides environment to deploy and undeploy the web applications.
> It will add middleware services only to deployed web applications.
> It provides environment to execute client side web resource programs at browser window.
> It is used to automate whole process of web application and web resource program executions.
> Web server will send the output to browser window as dynamic web page.

Web Container
It is a software application or a program which is use manage whole life cycle of web resource program from birth to death.
Servlet container manage whole life cycle of servlet program.
JSP container manage whole life cycle of jsp program.
some part of industry consider servlet container and jsp container are web containers.
Every server is designed to support servlet container and jsp container so we don't need to arrange them seperately.

Tomcat
=======

version	: 7.x
Creator	: James Duncan Davidson
Vendor	: Apache software foundation
Port No	: 8080
website	: www.apache.org
servlet container	: Catalina
Jsp container	: Jasper
Download	: https://drive.google.com/file/d/0B9rC21sL6v0tZFdVcmxZUDA0Tms/view?usp=drive_link&resourcekey=0-VXlB_IpeWqDWwdbr1baCyA

Tomcat install will ask following things.

1) Http Connector Port No
2) Adminstrator username and password
3) JRE location (parallel to JDK)
4) Tomcat Installation location

Tomcat is a not a container.It is a server containing servlet container and jsp container.

Before 6.x version tomcat is known as web server.From 6.x version onwards tomcat is also known as application server.

Installation of tomcat server
Right click to tomcat software --> yes ---> Next --> I Agree --> select Full -->
---> Next --->
Http connector port : 2525
adminstrative username : admin
password : admin
--> next --> Next --> Install.

Keeping Tomcat server from automatic mode to manual mode
services (view local services) --> select Apache Tomcat --> click to stop link -->
double click to Apache tomcat --> startup type: manual -->Apply -->ok.

Servlet
It is a dynamic web resource program which enhanced the functionality of web server or application server.

It is a java based dynamic web resource program which is used to create dynamic web pages.

It is a single instance multithread java based web resource program which is used to develop web applications.

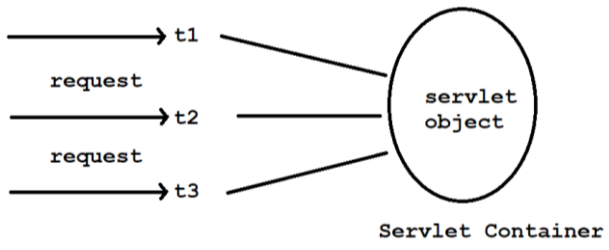

Diagram: servlet2.1

Servlet API
API is a collection of packges.
ex:
javax.servlet.*;
javax.servlet.http.*;

Important terminology

We have following important terminology.

1) javax.servlet.Servlet(I)

2) javax.servlet.GenericServlet(AC)

3) javax.servlet.http.HttpServlet(C)

First web application develop having servlet program as web resource program

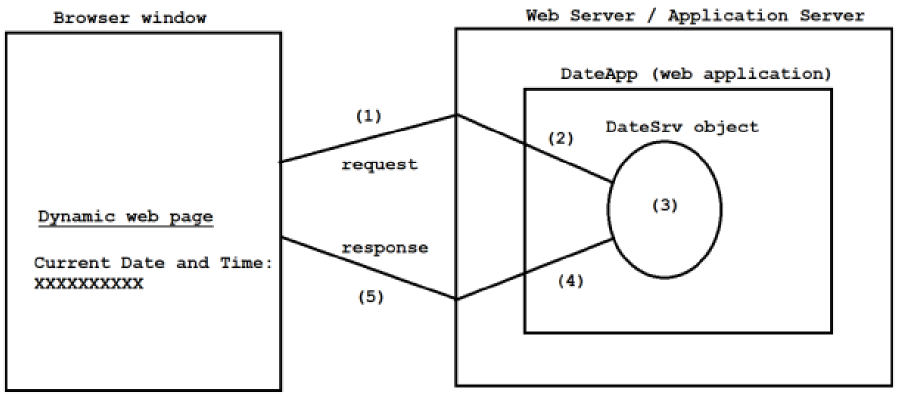

Diagram: servlet2.2

Deployment Directory structure

```
DateApp
|
|---Java Resources
|         |
          |------src
                    |
                    |--com.ihub.www
                              |
                              |---DateSrv.java
|
|---Web Content
|         |
          |------WEB-INF
                    |
                    |--web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

step1:

Launch eclipse IDE by choosing workspace location.

step2:

Create a dynamic web project i.e DateApp.

ex:

File --> new --> dynamic web project -->

project Name : DateApp
Dynamic web module version : 3.0 --> next --> next -->

select generate web.xml file(checkbox) --> finish.

step3:

Add "servlet-api.jar" in project build path.
ex:
right click to DateApp project --> build path --> configuration built path --> libraries --> Add external jars --> select servlet-api.jar file --> open -->ok .

step4:

Create a "com.ihub.www" package inside "src" folder.
ex:
Right click to src folder --> new --> package --> Name : com.ihub.www --> finish.

step5:

Create a "DateSrv.java" file inside "com.ihub.www" package.
ex:
right click to com.ihub.www pkg --> new --> class --> Name : DateSrv --> finish.

DateSrv.java

```
package com.ihub.www;

//ctrl+shift+o
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class DateSrv extends GenericServlet
{
	public void service(ServletRequest req,ServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");
		Date d=new Date();

		pw.println("<center><h1>Current Date And Time <br>"+d+"
</h1></center>");
		pw.close();
	}
}
```

step6:

Configure each servlet program in web.xml file.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
              <servlet-name>DateSrv</servlet-name>
              <servlet-class>com.ihub.www.DateSrv</servlet-class>
  </servlet>

  <servlet-mapping>
              <servlet-name>DateSrv</servlet-name>
              <url-pattern>/test</url-pattern>
  </servlet-mapping>



</web-app>
```

step7:

Add "Tomcat" server to eclipse IDE.

ex:

Window --> Preferences --> servers --> runtime Environments
--> click to Add button --> select Apache --> Apache Tomcat 7
---> Next --> select Tomcat installation directory --> finish
--> ok.

step8:

Run the dynamic project i.e DateApp.

ex:

Right click to DateApp --> run as --> run on server -->
select apache tomcat 7 --> finish.

step9:

Test the application by using below request url.

ex:

```
                              url pattern
                              |
        http://localhost:2525/DateApp/test
                |       |    |
         hostname   portno  webapplication
```

Note:
If there is a problem with web.xml file then we will get 404 Error.
If there is a problem with servlet program then we will get 500 Error.

Types of url patterns

Every servlet program will recognized with the help of URL pattern only.
URL pattern will hide servlet name and technology name from the outsider for security reason.
Our servlet container, client, other web resource programs will recognized servlet program with url pattern.
We have three types of URL patterns.

1) Exact Match url pattern
2) Directory Match url pattern
3) Extension Match url pattern

Every server is designed to support these three types of url patterns.

1) Exact Match url pattern

It will starts with '/' forward slash followed by a name.
ex:

<u>web.xml</u>

```
<url-pattern>/test</url-pattern>
```

<u>Request url</u>

```
http://localhost:2525/DateApp/test  (valid)
http://localhost:2525/DateApp/best  (invalid)
http://localhost:2525/DateApp/a/test  (invalid)
```

2) Directory Match url pattern

It will starts with '/' forward slash and ends with '*' symbol.
ex:

<u>web.xml</u>

```
<url-pattern>/x/y/*</url-pattern>
```

<u>Request url</u>

```
http://localhost:2525/DateApp/test  (invalid)
http://localhost:2525/DateApp/x/y/z  (valid)
http://localhost:2525/DateApp/x/y/z/test  (valid)
http://localhost:2525/DateApp/y/x/z  (invalid)
```

3) Extension Match url pattern

It will starts with '*' symbol followed by a extension.
ex:

<u>web.xml</u>

```
<url-pattern>*.do</url-pattern>
```

<u>Request url</u>

```
http://localhost:2525/DateApp/test  (invalid)
http://localhost:2525/DateApp/test.do (valid)
```

http://localhost:2525/DateApp/x/y/z/test (invalid)
http://localhost:2525/DateApp/y/x/z.do (valid)

MIME Types

MIME stands for Multipurpose Internet Mail Extension.
MIME describes in how many formats we can display our output in servlet.
We have following formats to display the output in servlet.

1) text/html
It is used to display the output in html format.

2) text/xml
It is used to display the output in xml format.

3) application/ms-word
It is used to display the output in word format.

4) application/vnd.ms-excel
It is used to display the output in excel format.

Deployment Directory Structure

```
MIMEApp
|
|---Java Resources
|         |
          |-----src
                    |
                    |---com.ihub.www
                              |
                              |---TestSrv1.java
                              |---TestSrv2.java
                              |---TestSrv3.java
                              |---TestSrv4.java
|
|---Web Content
          |
          |----WEB-INF
                    |
                    |---web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

If web application contains four servlets then we need to configure each servlet program in web.xml file with multiple <servlet> and <servlet-mapping> tags.

TestSrv1.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestSrv1 extends GenericServlet
{
	public void service(ServletRequest req,ServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		pw.println("<table border='1'>");
		pw.println("<tr><th>SNO</th><th>SNAME</th><th>SADD</th></tr>");
		pw.println("<tr><td>101</td><td>Alan</td><td>USA</td></tr>");
		pw.println("<tr><td>102</td><td>Jose</td><td>UAE</td></tr>");
		pw.println("<tr><td>103</td><td>Nelson</td><td>UK</td></tr>");
		pw.println("</table>");

		pw.close();
	}
}
```

TestSrv2.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestSrv2 extends GenericServlet
{
	public void service(ServletRequest req,ServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/xml");

		pw.println("<table border='1'>");
		pw.println("<tr><th>SNO</th><th>SNAME</th><th>SADD</th></tr>");
		pw.println("<tr><td>101</td><td>Alan</td><td>USA</td></tr>");
		pw.println("<tr><td>102</td><td>Jose</td><td>UAE</td></tr>");
		pw.println("<tr><td>103</td><td>Nelson</td><td>UK</td></tr>");
		pw.println("</table>");

		pw.close();
	}
}
```

TestSrv3.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestSrv3 extends GenericServlet
{
	public void service(ServletRequest req,ServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("application/ms-word");

		pw.println("<table border='1'>");
		pw.println("<tr><th>SNO</th><th>SNAME</th><th>SADD</th></tr>");
		pw.println("<tr><td>101</td><td>Alan</td><td>USA</td></tr>");
		pw.println("<tr><td>102</td><td>Jose</td><td>UAE</td></tr>");
		pw.println("<tr><td>103</td><td>Nelson</td><td>UK</td></tr>");
		pw.println("</table>");

		pw.close();
	}
}
```

TestSrv4.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class TestSrv4 extends GenericServlet
{
	public void service(ServletRequest req,ServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("application/vnd.ms-excel");

		pw.println("<table border='1'>");
		pw.println("<tr><th>SNO</th><th>SNAME</th><th>SADD</th></tr>");
		pw.println("<tr><td>101</td><td>Alan</td><td>USA</td></tr>");
		pw.println("<tr><td>102</td><td>Jose</td><td>UAE</td></tr>");
		pw.println("<tr><td>103</td><td>Nelson</td><td>UK</td></tr>");
		pw.println("</table>");

		pw.close();
	}
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
	<servlet-name>TestSrv1</servlet-name>
	<servlet-class>com.ihub.www.TestSrv1</servlet-class>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv1</servlet-name>
          <url-pattern>/html</url-pattern>
  </servlet-mapping>

  <servlet>
          <servlet-name>TestSrv2</servlet-name>
          <servlet-class>com.ihub.www.TestSrv2</servlet-class>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv2</servlet-name>
          <url-pattern>/xml</url-pattern>
  </servlet-mapping>


  <servlet>
          <servlet-name>TestSrv3</servlet-name>
          <servlet-class>com.ihub.www.TestSrv3</servlet-class>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv3</servlet-name>
          <url-pattern>/word</url-pattern>
  </servlet-mapping>


  <servlet>
          <servlet-name>TestSrv4</servlet-name>
          <servlet-class>com.ihub.www.TestSrv4</servlet-class>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv4</servlet-name>
          <url-pattern>/excel</url-pattern>
  </servlet-mapping>

</web-app>
```

request url

```
          http://localhost:2525/MIMEApp/html
          http://localhost:2525/MIMEApp/xml
          http://localhost:2525/MIMEApp/word
          http://localhost:2525/MIMEApp/excel
```

Types of communication

We can communicate to servlet program in three ways.

1) Browser to servlet communication
2) HTML to servlet communication
3) Servlet to servlet communication

In browser to servlet communication we need to type our request url in browser address bar.But typing request url in browser address is quit complex.

To overcome this limitation we need to use HTML to servlet communication.

In html to servlet communication we can forward the request to servlet program using html based hyperlinks and form pages.

A request which is generated by using hyperlink does not carry the data.

But a request which is generated by using form page will carry the data.

In html based hyperlink to servlet communication we need to type our request url as href url.

ex:

```
        <a href="http://localhost:2525/DateApp/test"> click Here </a>
```

In html based form page to servlet communication we need to type our request url as action url.

ex:

```
        <form action="http://localhost:2525/DateApp/test">
                -
                -
        </form>
```

Example application on Html based hyperlink to servlet communication

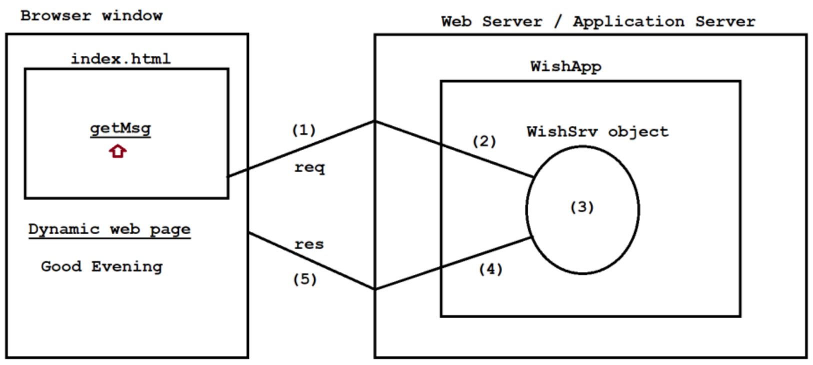

Diagram: servlet3.1

<u>Deployment Directory structure</u>

```
WishApp
|
|---Java Resource
        |
        |------src
                |
                |---com.ihub.www
                        |
                        |---WishSrv.java
```

```
|
|---Web Content
          |
          |----index.html
          |
          |----WEB-INF
                    |
                    |---web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.
It is never recommanded to extends a class with GenericServlet class because it will not give HTTP protocol features.
It is always recommanded to extends a class with HttpServlet class because it will give HTTP protocol features.

<u>index.html</u>

```html
<center>
          <h1>
                    <a href="test"> getMsg </a>
          </h1>
</center>
```

<u>web.xml</u>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

 <servlet>
          <servlet-name>WishSrv</servlet-name>
          <servlet-class>com.ihub.www.WishSrv</servlet-class>
 </servlet>
 <servlet-mapping>
          <servlet-name>WishSrv</servlet-name>
          <url-pattern>/test</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
          <welcome-file>index.html</welcome-file>
 </welcome-file-list>

</web-app>
```

TestSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Calendar;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class WishSrv extends HttpServlet
{
	public void service(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		Calendar c=Calendar.getInstance();
		//convert time to 24 hours
		int h=c.get(Calendar.HOUR_OF_DAY);

		if(h<12)
			pw.println("<center><h1>Good Morning</h1></center>");
		else if(h<16)
			pw.println("<center><h1>Good afternoon</h1></center>");
		else if(h<20)
			pw.println("<center><h1>Good Evening</h1></center>");
		else
			pw.println("<center><h1>Good Night</h1></center>");

		pw.close();

	}
}
```

Request url

```
	http://localhost:2525/WishApp/test
```

Example application on HTML based form page to servlet communication

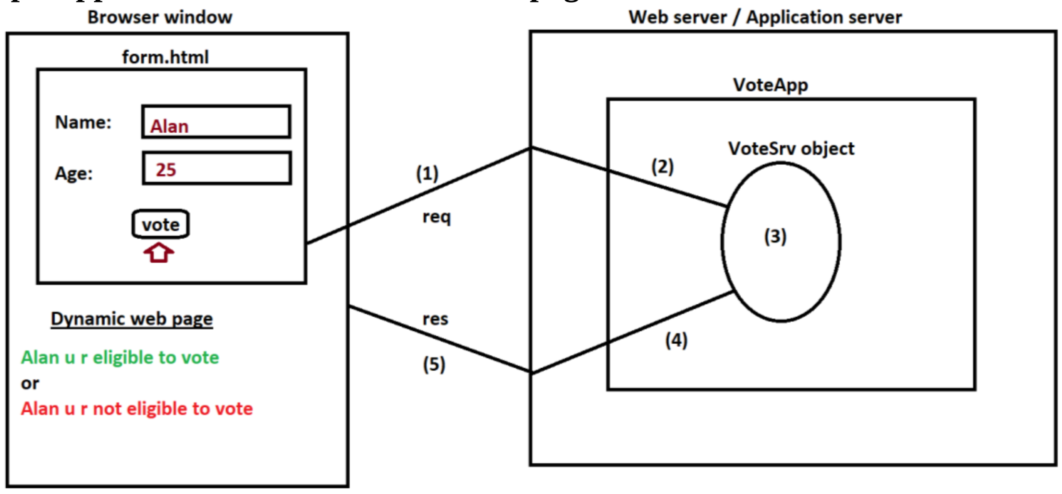

Diagram: servlet4.1

Deployment Directory structure

```
VoteApp
|
|----Java Resources
          |
          |-------src
                    |
                    |----com.ihub.www
                              |
                              |----VoteSrv.java
|----Web Content
|         |
          |---form.html
          |
          |----WEB-INF
                    |
                    |------web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.
We can send the request to servlet intwo methodologies.

1) GET Methodology
It will carry limited amount of data.
2) POST methodology
It will carry unlimited amount of data.

While working with HttpServlet class, it is never recommanded to work with service(-,-) method because it is not designed according HTTP protocol features. It is always recommanded to use doXxx(-,-) methods because they have designed according HTTP protocol features.

We have seven doXxx(-,-) methods as follow.
ex:

1) doGet(-,-)
2) doPost(-,-)
3) doHead(-,-)
4) doOption(-,-)
5) doTrace(-,-)
6) doDelete(-,-)
7) doPut(-,-)

prototype of doXxx(-,-)

```
protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
{
}
```

form.html

```
<form action="test" method="GET">
	Name: <input type="text" name="t1"/> <br>
	Age: <input type="text" name="t2"/> <br>
	<input type="submit" value="vote"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

 <servlet>
	<servlet-name>VoteSrv</servlet-name>
	<servlet-class>com.ihub.www.VoteSrv</servlet-class>
 </servlet>
 <servlet-mapping>
	<servlet-name>VoteSrv</servlet-name>
	<url-pattern>/test</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
	<welcome-file>form.html</welcome-file>
 </welcome-file-list>

</web-app>
```

VoteSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class VoteSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form data
		String name=req.getParameter("t1");
		String sage=req.getParameter("t2");

		//converting string age to int age
		int age=Integer.parseInt(sage);

		if(age>=18)
			pw.println("<center><h1 style='color:green'>"+name+" U r eligible to
vote</h1></center>");
		else
			pw.println("<center><h1 style='color:red'>"+name+" U r not eligible
to vote</h1></center>");

		pw.close();
	}
}
```

request url
----
	http://localhost:2525/VoteApp/

Q)What is the difference between GET and POST methodology?

GET	POST
It is a default methodology.	It is not a default methodology.
It will carry limited amount of data.	It will carry unlimited amount of data.
It sends the request fastly.	It sends the request bit slow.
It is not suitable for secure data.	It is suitable for secure data.
It is not suitable for encryption and file uplaoding.	It is suitable for encryption and file uploading.
To process get methodology we will use doGet(-,-) method.	To process post methodology we will use doPost(-,-) method.

Servlet to Database Communication

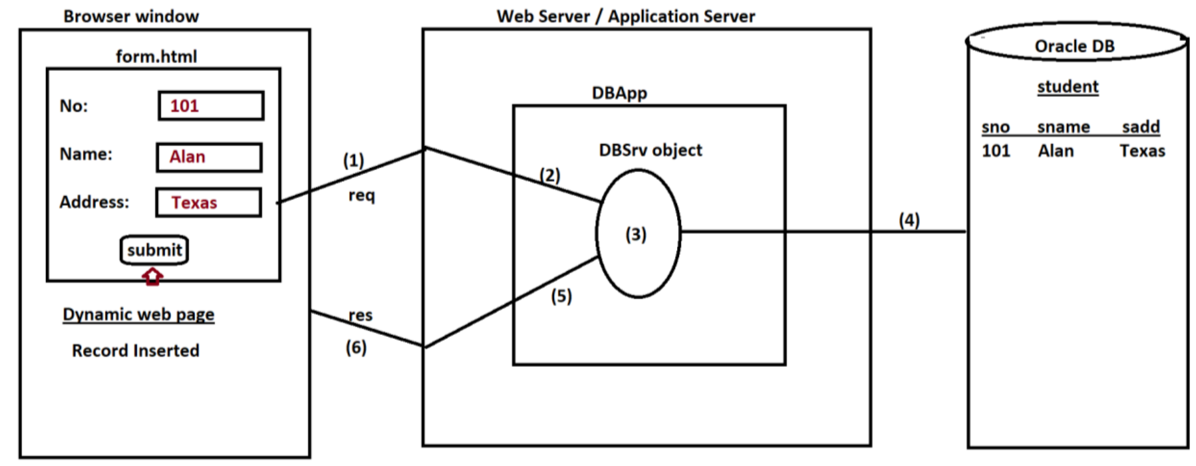

Diagram: servlet4.2

Deployment Directory structure

```
DBApp
|
|---Java Resources
          |
          |-----src
                    |
                    |----com.ihub.www
                              |
                              |----DBSrv.java
|
|---Web Content
          |
          |-----form.html
          |
          |-----WEB-INF
                    |
                    |-----web.xml
                    |
                    |------lib
                              |
                              |---ojdbc14.jar
```

Note:
In above application we need to "servlet-api.jar" and "ojdbc14.jar" file in project build path.
Copy and paste "ojdbc14.jar" file in "WEB-INF/lib" folder seperately.

<u>student table</u>

```
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
```

<u>form.html</u>

```
<form action="test" method="GET">
        No: <input type="text" name="t1"/> <br>
        Name: <input type="text" name="t2"/> <br>
        Address: <input type="text" name="t3"/> <br>
        <input type="submit" value="submit"/>
</form>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

 <servlet>
        <servlet-name>DBSrv</servlet-name>
        <servlet-class>com.ihub.www.DBSrv</servlet-class>
 </servlet>
 <servlet-mapping>
        <servlet-name>DBSrv</servlet-name>
        <url-pattern>/test</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
        <welcome-file>form.html</welcome-file>
 </welcome-file-list>

</web-app>
```

DBSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DBSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form data
		String sno=req.getParameter("t1");
		int no=Integer.parseInt(sno);
		String name=req.getParameter("t2");
		String add=req.getParameter("t3");

		//insert form data to database
		Connection con=null;
		PreparedStatement ps=null;
		int result=0;
		String qry=null;
		try
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");
	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system",
"admin");
			qry="insert into student values(?,?,?)";
			ps=con.prepareStatement(qry);
			//set the values
			ps.setInt(1,no);
			ps.setString(2,name);
			ps.setString(3,add);
```

```
				//execute
				result=ps.executeUpdate();

				if(result==0)
					pw.println("<center>No Record Inserted</center>");
				else
					pw.println("<center>Record Inserted</center>");

				ps.close();
				con.close();

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

Request url

	http://localhost:2525/DBApp/

Form Validation
The process of checking format and pattern of form data is called form validation and such logic is called form validation logic.

We can perform form validation in two ways.
1) Client side form validation
	Validation which is performed at client side is called client side form validation.
2) Server side form validation
	Validation which is performed at server side is called server side form validation.

Deployment Directory structure

```
ValidationApp
|
|----Java Resources
        |
        |-------src
                |
                |---com.ihub.www
                        |
                        |-----FormSrv.java
```

```
|
|----Web Content
        |
        |------form.html
        |
        |------validation.js
        |
        |------WEB-INF
                |
                |-----web.xml
```
Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>form.html</u>
```
<!DOCTYPE html>
<html>
        <head>
                <!--  add javascript file -->
                <script type="text/javascript" src="validation.js"></script>

        </head>
        <body>
                <form   name="myform"   action="test"   method="GET"   onsubmit="return
validate()">

                                Name: <input type="text" name="t1"/> <br>

                                Age: <input type="text" name="t2"/> <br>

                                <input type="hidden" name="vflag" value="no"/> <br>

                                <input type="submit" value="submit"/>
                </form>
        </body>
</html>
```

<u>web.xml</u>
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
 <servlet>
        <servlet-name>FormSrv</servlet-name>
        <servlet-class>com.ihub.www.FormSrv</servlet-class>
 </servlet>
 <servlet-mapping>
        <servlet-name>FormSrv</servlet-name>
        <url-pattern>/test</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
        <welcome-file>form.html</welcome-file>
 </welcome-file-list>
 </web-app>
```

<u>validation.js</u>

```
function validate()
{

        var name=document.myform.t1.value;
        var age=document.myform.t2.value;
        document.myform.vflag.value="yes";

        if(name=="")
        {
                alert("Name is mandatory");
                document.myform.t1.focus();
                return false;
        }
        if(age=="")
        {
                alert("Age is mandatory");
                document.myform.t2.focus();
                return false;
        }
        else
        {
                if(isNaN(age))
                {
                        alert("Age must be numeric ");
                        document.myform.t2.value="";
                        document.myform.t2.focus();
                        return false;
                }
        }

        return true;
}
```

FormSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FormSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form data
		String name=req.getParameter("t1");
		String sage=req.getParameter("t2");
		String status=req.getParameter("vflag");
		int age=0;

		if(status.equals("no"))
		{
			if(name=="" || name.length()==0 || name==null)
			{
				pw.println("<center>Name is mandatory</center>");
				return;
			}
			if(sage=="" || sage.length()==0 || sage==null)
			{
				pw.println("<center>Age is mandatory</center>");
				return;
			}
			else
			{
				try
				{
					age=Integer.parseInt(sage);
				}
				catch(NumberFormatException nfe)
```

```
                                {
                                        pw.println("<center>Age must be numeric</center>");
                                        return;
                                }
                        }
                }

                if(status.equals("yes"))
                {
                        age=Integer.parseInt(sage);
                }

                if(age<18)
                        pw.println("<center>U r not eligible to vote</center>");
                else
                        pw.println("<center>U r eligible to vote</center>");
                pw.close();
        }
}
```

<u>Request url</u>

    http://localhost:2525/ValidationApp/

File Uploading

The process of capturing a file from client machine file system and storing in a server machine file system is called file uploading and reverse is called file downloading.
While dealing with matrimonial applications,job portal applications and profile management applications we need to upload and download a file.
There is no specific API in Servlet to perform file uploading.
We need to take the support of third party API called JAVAZOOM API.
JAVAZOOM API comes in zip format and once if we extract then we will get three jar files.
ex:

```
        uploadbean.jar (main jar file)
        struts.jar     (dependent jar file)
        cos.jar        (dependent jar file)
```

We can take file component in a form page as follow.
ex:

```
        <input type="file" name="f1"/>
```

<u>JAVAZOOM API link</u>

https://drive.google.com/file/d/1LB0WSJvSCCVOgz7xNwyuYtmy_0_TfJzq/view?usp=sharing

Deployment Directory structure

```
UploadApp
|
|----Java Resources
|         |
          |-----src
                    |
                    |----com.ihub.www
                              |
                              |---TestSrv.java
|
|----Web Content
|         |
          |-----form.html
          |
          |-----WEB-INF
                    |
                    |------web.xml
                    |
                    |-------lib
                              |
                              |----uploadbean.jar
                              |----struts.jar
                              |----cos.jar
```

Note:
In above application we need to add "servlet-api.jar" and "uploadbean.jar" file in project build path.
copy and paste javazoom api jar files inside "WEB-INF/lib" folder seperately.

form.html

```
<form action="test" method="POST" enctype="multipart/form-data">

          File1: <input type="file" name="f1"/> <br>

          File2: <input type="file" name="f2"/> <br>

          <input type="submit" value="upload"/>

</form>
```

web.xml

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

```
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
        <servlet-name>TestSrv</servlet-name>
        <servlet-class>com.ihub.www.TestSrv</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>TestSrv</servlet-name>
        <url-pattern>/test</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>TestSrv.java</u>

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import javazoom.upload.MultipartFormDataRequest;
import javazoom.upload.UploadBean;

public class TestSrv extends HttpServlet
{
        protected void doPost(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
                PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                //file uploading
                try
                {
                        UploadBean ub=new UploadBean();
                        ub.setFolderstore("C:\\B24");
                        ub.setOverwrite(false);

                        MultipartFormDataRequest                              nreq=new
MultipartFormDataRequest(req);
                        ub.store(nreq);

                        pw.println("<center>Files are uploaded successfully</center>");
                }
                catch(Exception e)
                {
                        pw.println(e);
                }
                pw.close();
        }
}
```

<u>Request url</u>

    http://localhost:2525/UploadApp/

Servlet Filters

Filter is an object which is executed at the time of preprocessing and post processing of the request.

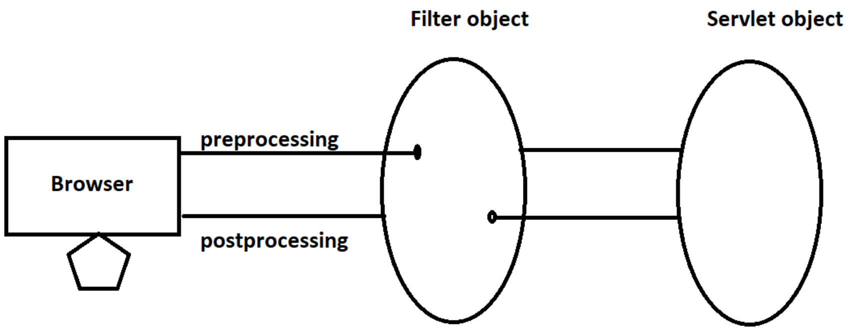

Diagram: servlet6.1

The main purpose of filters are

1) We can count number of request coming to the application
2) To perform validations.
3) To perform encryption and Decryption.

Like Servlet, Filter is having it's own Filter API.

The javax.servlet package contains thre interfaces of Filter API.

1)Filter
2)FilterChain
3)FilterConfig

1) Filter Interface
For creating any filter, we must and should implements the Filter interface.
Filter interface provides the following 3 life cycle methods for filter.
i) public void init(FilterConfig config)
IT is used to initialize the filter.
It invokes only once .

ii) public void doFilter(HttpServletRequest req,HttpServletResponse res,FilterChain chain)
This method is invoked every time when user request to any resources to which the filter is mappend.
IT is used to perform filtering task.

iii) public void destroy()
This method is invoked only once when filter is taken out of the service.

2) FilterChain
It is responsible to invoke the next filter or resource in the chain.
FilterChain contains only one method.
i) public void doFilter(HttpServletRequest req,HttpServletResponse res)
It passes the control to the next filter or resource.

3) FilterConfig
For every filter our servlet container creates FilterConfig object.
It is one per filter.

Deployment Directory structure

```
FilterApp
|
|----Java Resources
        |
        |------src
                |
                |----com.ihub.www
                        |
                        |---MyFilter.java
                        |---MyServlet.java
|
|----Web Content
        |
        |------index.html
        |
        |------WEB-INF
                |
                |------web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>index.html</u>

```html
<center>
	<h1>
		<a href="test"> clickMe </a>
	</h1>
</center>
```

<u>web.xml</u>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
	<servlet-name>MyServlet</servlet-name>
	<servlet-class>com.ihub.www.MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
	<servlet-name>MyServlet</servlet-name>
	<url-pattern>/test</url-pattern>
  </servlet-mapping>

  <filter>
	<filter-name>MyFilter</filter-name>
	<filter-class>com.ihub.www.MyFilter</filter-class>
  </filter>
  <filter-mapping>
	<filter-name>MyFilter</filter-name>
	<url-pattern>/test</url-pattern>
  </filter-mapping>

  <welcome-file-list>
	<welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

MyFilter.java
```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements Filter
{
	@Override
	public void init(FilterConfig config) throws ServletException {
		// TODO Auto-generated method stub

	}


	@Override
	public void doFilter(ServletRequest req, ServletResponse res,
			FilterChain chain) throws IOException, ServletException {

		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		pw.println("<center>Filter Invoked Before</center><br>");
		chain.doFilter(req,res);
		pw.println("<center>Filter Invoked After</center><br>");
	}


	@Override
	public void destroy() {
		// TODO Auto-generated method stub

	}

}
```

MyServlet.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		pw.println("<center>Servlet Executed</center><br>");
	}
}
```

Request url

	http://localhost:2525/FilterApp/test

Servlet Life cycle methods

We have three life cycle methods in servlets.

1) public void init(ServletConfig config)throws ServletException

It is used for instantiation event.
This method will execute just before servlet object creation.

2) public void service(ServletRequest req,ServletResponse res) throws ServletException,IOException

It is used for request arrival event.
This method will execute when request goes to servlet program.

3) public void destroy()

It is used for destruction event.
This method will execute just before servlet object destruction.

Deployment Directory structure

```
LifeCycleApp
|
|----Java Resources
|                    |
                     |----src
                                   |
                                   |----com.ihub.www
                                                     |
                                                     |-----TestSrv.java
|
|----Web Content
                     |
                     |---index.html
                     |
                     |-----WEB-INF
                                        |
                                        |-----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

index.html

```
<center>
        <h1>
                <a href="test"> click Here  </a>
        </h1>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
        <servlet-name>TestSrv</servlet-name>
        <servlet-class>com.ihub.www.TestSrv</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>TestSrv</servlet-name>
        <url-pattern>/test</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
          <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

TestSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;

public class TestSrv extends HttpServlet
{
                public void init(ServletConfig config)throws ServletException
                {
                        System.out.println("init-method");
                }

                public void service(ServletRequest req,ServletResponse res)throws
ServletException,IOException
                {
                                PrintWriter pw=res.getWriter();
                                res.setContentType("text/html");

                                pw.println("<center>Service method is called</center>");
                                pw.close();
                                System.out.println("service-method");
                }
                public void destroy()
                {
                        System.out.println("destroy-method");
                }
}
```

Request url

```
        http://localhost:2525/LifeCycleApp/test
```

ServletConfig object

ServletConfig is an interface which is present in javax.servlet package.
ServletConfig object is created by the web container for every servlet.
Servletconfig object is used to read configuration information from web.xml file.

We can create ServletCofig object as follow.
ex:

```
        ServletConfig config=getServletConfig();
```

ServletConfig interface contains following four methods.

1) public String getInitParameter(String name);
It will return parameter value based on specified parameter name.

2) public Enumeration getInitParameterNames();
It will return enumeration of all initialized parameter names.

3) public ServletContext getServletContext();
It will return ServletContext object.

4) public String getServletName();
It will return Servlet name.

Deployment Directory structure

```
ConfigApp
|
|----Java Resources
|               |
                |-------src
                        |
                        |----com.ihub.www
                                        |
                                        |-----TestSrv.java
|
|----Web Content
                |
                |---index.html
                |
                |-----WEB-INF
                                        |
                                        |-----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

index.html

```
<center>
	<h1>
		<a href="test"> click Here </a>
	</h1>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
	<servlet-name>TestSrv</servlet-name>
	<servlet-class>com.ihub.www.TestSrv</servlet-class>
	<init-param>
		<param-name>driver</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:XE</param-value>
	</init-param>
  </servlet>


  <servlet-mapping>
	<servlet-name>TestSrv</servlet-name>
	<url-pattern>/test</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
	<welcome-file>index.html</welcome-file>
  </welcome-file-list>


</web-app>
```

TestSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		ServletConfig config=getServletConfig();

		pw.println(config.getInitParameter("driver")+"<br>");
		pw.println(config.getInitParameter("url")+"<br>");

		Enumeration<String> e=config.getInitParameterNames();
		while(e.hasMoreElements())
		{
			String s=e.nextElement();
			pw.println(s+"<br>");
		}

		pw.println(config.getServletName()+"<br>");
		pw.close();
	}
}
```

Request url

```
	http://localhost:2525/ConfigApp/test
```

ServletContext object

ServletContext is an interface which is present in javax.servlet package.
ServletContext object is created by the web container for every web application.
Servletcontext object is used to read configuration information from web.xml file which is global.

We can create ServletContext object as follow.
ex:

```
ServletContext context=getServletContext();
```

ServletContext contains following methods.

1) public String getInitParameter(String name);
It will return parameter value based on specified parameter name.

2) public Enumeration getInitParameterNames();
It will return enumeration of all initialized parameter names.

Deployment Directory structure

```
ContextApp
|
|----Java Resources
|            |
             |----src
                      |
                      |----com.ihub.www
                                    |
                                    |-----TestSrv.java
|
|----Web Content
             |
             |---index.html
             |
             |-----WEB-INF
                           |
                           |-----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

index.html

```
<center>
        <h1>
                <a href="test">Click Here </a>
        </h1>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
        <servlet-name>TestSrv</servlet-name>
        <servlet-class>com.ihub.www.TestSrv</servlet-class>
  </servlet>

  <servlet-mapping>
        <servlet-name>TestSrv</servlet-name>
        <url-pattern>/test</url-pattern>
  </servlet-mapping>


        <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:XE</param-value>
        </context-param>


</web-app>
```

}

TestSrv.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		ServletContext context=getServletContext();


		pw.println(context.getInitParameter("driver")+"<br>");
		pw.println(context.getInitParameter("url")+"<br>");

		Enumeration<String> e=context.getInitParameterNames();
		while(e.hasMoreElements())
		{
			String s=e.nextElement();
			pw.println(s+"<br>");
		}


		pw.close();


	}
}
```

Request url

```
	http://localhost:2525/ContextApp
```

Servlet to Servlet Communication

Servlet to Servlet communication is also known as servlet chaining.

Servlet to servlet communication is possible by using three ways.

1)Forwarding the request
2)Including the response
3)Send Redirection

1) Forwarding the request

In forwarding the request, output of source servlet program will be discarded and only output of destination servlet program goes to browser window as dynamic response.
To forward the request we need to use RequestDispatcher object.
RequestDispatcher is an interface which is present in javax.servlet package.
We can create RequestDispatcher object as follow.
ex:

```
	RequestDispatcher rd=req.getRequestDispatcher("url");
	rd.forward(req,res);
```

2) Including the response

In cluding the response, output of source servlet program and output of destination servlet program combinely goes to browser window as dynamic response.
For including the response we need to use RequestDispatcher object.
RequestDispatcher is an interface which is present in javax.servlet package.
We can create RequestDispatcher object as follow.
ex:

```
	RequestDispatcher rd=req.getRequestDispatcher("url");
	rd.include(req,res);
```

<u>Deployment Directory structure</u>

```
STSApp1
|
|----Java Resources
	|
	|-----src
		|
		|---com.ihub.www
			|
			|---TestSrv1.java
			|---TestSrv2.java
|
|----Web Content
	|
	|-----form.html
	|
	|------WEB-INF
		|
		|----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>form.html</u>

```
<form action="test1">
        <table align="center">

                <tr>
                        <td>UserName:</td>
                        <td><input type="text" name="t1"/></td>
                </tr>
                <tr>
                        <td>Password:</td>
                        <td><input type="password" name="t2"/></td>
                </tr>
                <tr>
                        <td><input type="reset" value="reset"/></td>
                        <td><input type="submit" value="submit"/></td>
                </tr>

        </table>
</form>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
        <servlet-name>TestSrv1</servlet-name>
        <servlet-class>com.ihub.www.TestSrv1</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>TestSrv1</servlet-name>
        <url-pattern>/test1</url-pattern>
  </servlet-mapping>

  <servlet>
        <servlet-name>TestSrv2</servlet-name>
        <servlet-class>com.ihub.www.TestSrv2</servlet-class>
  </servlet>
  <servlet-mapping>
		<servlet-name>TestSrv2</servlet-name>
		<url-pattern>/test2</url-pattern>
  </servlet-mapping>

		<welcome-file-list>
			<welcome-file>form.html</welcome-file>
		</welcome-file-list>

</web-app>
```

<u>TestSrv1.java</u>

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv1 extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form data
		String name=req.getParameter("t1");
		String pwd=req.getParameter("t2");

		if(pwd.equals("admin"))
		{
			RequestDispatcher rd=req.getRequestDispatcher("test2");
			rd.forward(req,res);
		}
		else
		{
			pw.println("<center><b style='color:red'>Sorry! Incorrect username or
password</b></center>");
			RequestDispatcher rd=req.getRequestDispatcher("form.html");
```

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

            pw.close();
        }
}
```

<u>TestSrv2.java</u>

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv2 extends HttpServlet
{
        protected void doGet(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
        {
                PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                pw.println("<center>Login Successfully!!!</center>");

                pw.close();
        }
}
```

<u>Request url</u>

```
        http://localhost:2525/STSApp1/
```

3) Send Redirection

It is used to forward the request to web application which is present in same server or different server.

Send Redirection by using sendRedirect() method HttpServletResponse object.

ex:

```
        res.sendRedirect("url");
```

Send redirection will work inside the server and outside the server.

It always sends a new request.

It uses browser window to send the request.

Deployment Directory structure

```
STSApp2
|
|----Java Resources
          |
          |-----src
                    |
                    |---com.ihub.www
                              |
                              |---TestSrv.java
|
|----Web Content
          |
          |-----index.html
          |
          |------WEB-INF
                    |
                    |----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

index.html

```
<center>
          <h1>
                              <a href="test?t1=flights"> Flights </a>  <br><br>
                              <a href="test?t1=hotels"> Hotels </a>   <br><br>
                              <a href="test?t1=railways"> Trains </a>   <br><br>
          </h1>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <servlet>
          <servlet-name>TestSrv</servlet-name>
          <servlet-class>com.ihub.www.TestSrv</servlet-class>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv</servlet-name>
          <url-pattern>/test</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
        <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>TestSrv.java</u>

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv extends HttpServlet
{
        protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
                PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                String val=req.getParameter("t1");

                res.sendRedirect("https://www.makemytrip.com/"+val);

                pw.close();

        }
}
```

Request url
----
        http://localhost:2525/STSApp2/

Q) How to enable <load-on-startup> and what happens if we enable <load-on-startup>?

We can enable <load-on-startup> inside web.xml file.

If we enable <load-on-startup> then our servlet container will create servlet object during the server startup or during the deployment of web application.

web.xml

```
<web-app>
          <servlet>
                    <servlet-name>TestSrv2</servlet-name>
                    <servlet-class>com.ihub.www.TestSrv2</servlet-class>
                    <load-on-startup>1</load-on-startup>
          </servlet>
          <servlet-mapping>
                    <servlet-name>TestSrv2</servlet-name>
                    <url-pattern>/test2</url-pattern>
          </servlet-mapping>
</web-app>
```

Stateless Behaviour of web application

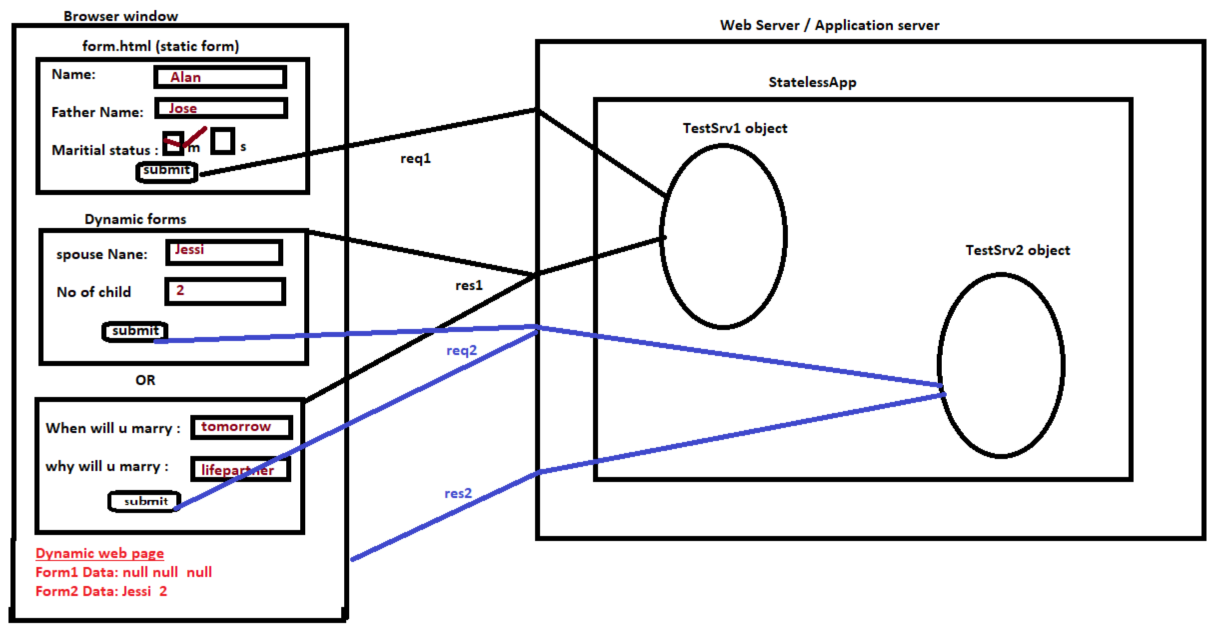

Diagram: servlet9.1

In above diagram demostrate stateless behaviour of web application.

Stateless web application means while working with current request we can'taccess previous request data.

Our HTTP protocol is stateless which makes our web application also stateless.

To overcome this limitation we need to use Session Tracking.

Deployment Directory structure

```
StatelessApp
|
|---Java Resources
|         |
          |------src
                   |
                   |---com.ihub.www
                            |
                            |----TestSrv1.java
                            |----TestSrv2.java
|---Web Content
|         |
          |------form.html
          |
          |------WEB-INF
                   |
                   |---web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

form.html

```
<form action="test1">
        Name: <input type="text" name="t1"/> <br>
        Father Name : <input type="text" name="t2"/> <br>
        Maritial Status :
        <input type="checkbox" name="t3" value="married"/> MARRIED
        <input type="checkbox" name="t3" value="single"/> SINGLE
        <br>
        <input type="submit" value="submit"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
        <servlet-name>TestSrv1</servlet-name>
        <servlet-class>com.ihub.www.TestSrv1</servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv1</servlet-name>
          <url-pattern>/test1</url-pattern>
  </servlet-mapping>

  <servlet>
          <servlet-name>TestSrv2</servlet-name>
          <servlet-class>com.ihub.www.TestSrv2</servlet-class>
          <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
          <servlet-name>TestSrv2</servlet-name>
          <url-pattern>/test2</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
          <welcome-file>form.html</welcome-file>
  </welcome-file-list>


</web-app>
```

TestSrv1.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv1 extends HttpServlet
{
        protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
                PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                //reading form1  data

                String name=req.getParameter("t1");
```

```
		String fname=req.getParameter("t2");
		String ms=req.getParameter("t3");

		if(ms.equals("married"))
		{
			pw.println("<form action='test2'>");
			pw.println("Spouse Name :<input type='text' name='f2t1'/> <br>");
			pw.println("No of Children: <input type='text' name='f2t2'/> <br>");
			pw.println("<input type='submit' value='submit'/>");
			pw.println("</form>");
		}
		else
		{
			pw.println("<form action='test2'>");
			pw.println("When will u marry :<input type='text' name='f2t1'/>
<br>");
			pw.println("Why will u marry: <input type='text' name='f2t2'/> <br>");
			pw.println("<input type='submit' value='submit'/>");
			pw.println("</form>");
		}
		pw.close();
	}
}
```

TestSrv2.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TestSrv2 extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form1  data
```

```
                String name=req.getParameter("t1");
                String fname=req.getParameter("t2");
                String ms=req.getParameter("t3");

                //reading form2 data
                String val1=req.getParameter("f2t1");
                String val2=req.getParameter("f2t2");

                pw.println("Form 1 Data :"+name+" "+fname+" "+ms+"<br>");
                pw.println("Form 2 Data :"+val1+" "+val2+"<br>");

                pw.close();
        }
}
```

Request url

http://localhost:2525/StatelessApp/

Session

The process of continue and related operations perform on web application with multiple request and response is called session.

ex:

login to facebook and logout from facebook is one session.

starting of java class and ending of java class is one session.

Session Tracking / Session Management

Session tracking makes our web application as statefull web application even thoughour HTTP protocol is stateless.

In stateless web application, no web resource program can access previous request data while processing the current request during a session.

In statefull web application , all web resource programs can access previous request data while processing the current request during a session.

Session tracking can be perform in four techniques.

1) Using hidden box fields
2) HttpCookies
3) HttpSession with Cookies
4) URL Rewriting

3) HttpSesion with Cookies.

HttpSession is an interface which is present in javax.servlet package.

HttpSession always create a session ID for every request to identify where enduser is a existing user or old user.

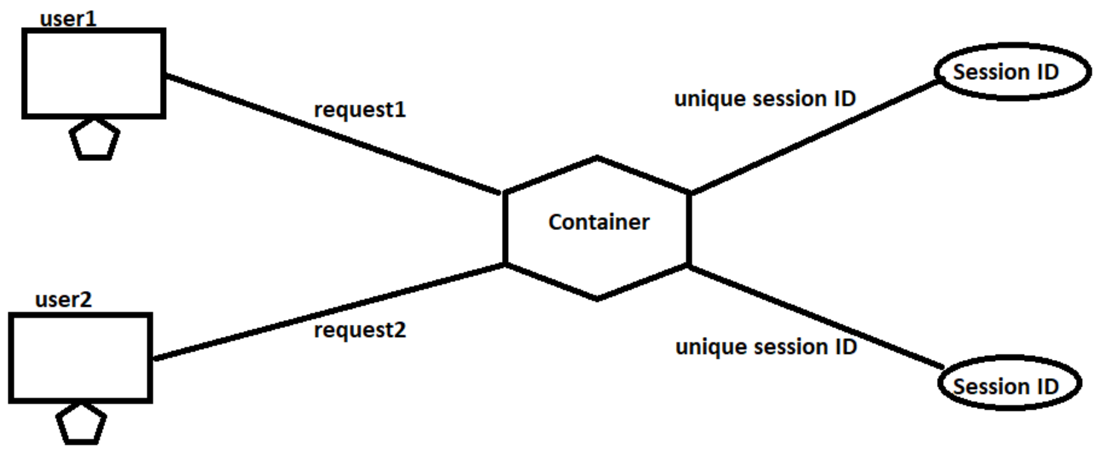

Diagram: servlet9.2

HttpSession object is to perform following things.

1) Bind the object
2) It will manipulate the data present HttpSession.

<u>Deployment Directory structure</u>

```
SessionTrackingApp
|
|----Java Resources
|         |
          |------src
                    |
                    |----com.ihub.www
                              |
                              |----TestSrv1.java
                              |----TestSrv2.java
|
|----Web Content
|         |
          |-------form.html
          |
          |-------WEB-INF
                    |
                    |----web.xml
```

Note:
In above application , we need to add "servlet-api.jar" file in project build path.

<u>form.html</u>

```
<form action="test1" method="GET">

          Name: <input type="text" name="t1"/> <br>
```

```
	Father Name : <input type="text" name="t2"/> <br>

	Maritial Status :
	<input type="checkbox" name="t3" value="married"/>MARRIED
	<input type="checkbox" name="t3" value="single"/>SINGLE
	<br>

	<input type="submit" value="submit"/>

</form>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

 <servlet>
	<servlet-name>TestSrv1</servlet-name>
	<servlet-class>com.ihub.www.TestSrv1</servlet-class>
 </servlet>
 <servlet-mapping>
	<servlet-name>TestSrv1</servlet-name>
	<url-pattern>/test1</url-pattern>
 </servlet-mapping>

 <servlet>
	<servlet-name>TestSrv2</servlet-name>
	<servlet-class>com.ihub.www.TestSrv2</servlet-class>
 </servlet>
 <servlet-mapping>
	<servlet-name>TestSrv2</servlet-name>
	<url-pattern>/test2</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
	<welcome-file>form.html</welcome-file>
 </welcome-file-list>


</web-app>
```

TestSrv1.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class TestSrv1 extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		String name=req.getParameter("t1");
		String fname=req.getParameter("t2");
		String ms=req.getParameter("t3");

		HttpSession session=req.getSession(true);
		session.setAttribute("pname",name);
		session.setAttribute("pfname", fname);
		session.setAttribute("pms", ms);

		if(ms.equals("married"))
		{
			pw.println("<form action='test2'>");
			pw.println("spouse Name : <input type='text' name='f2t1'/> <br>");
			pw.println("No of Child : <input type='text' name='f2t2'/> <br>");
			pw.println("<input type='submit' value='submit'/>");
			pw.println("</form>");
		}
		else
		{
			pw.println("<form action='test2'>");
			pw.println("When will u marry : <input type='text' name='f2t1'/>
<br>");
			pw.println("Why will u marry : <input type='text' name='f2t2'/>
<br>");
```

```
                        pw.println("<input type='submit' value='submit'/>");
                        pw.println("</form>");
                }
                pw.close();
        }
}
```

TestSrv2.java

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class TestSrv2 extends HttpServlet
{
        protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
                PrintWriter pw=res.getWriter();
                res.setContentType("text/html");

                //reading form 1 data
                HttpSession session=req.getSession(false);
                String name=(String)session.getAttribute("pname");
                String fname=(String)session.getAttribute("pfname");
                String ms=(String)session.getAttribute("pms");
                //reading form 2 data
                String val1=req.getParameter("f2t1");
                String val2=req.getParameter("f2t2");

                pw.println("Form 1 data :"+name+" "+fname+" "+ms+"<br>");
                pw.println("Form 2 data :"+val1+" "+val2+"<br>");

                pw.close();
        }
}
```

Request url

```
        http://localhost:2525/SessionTrackingApp/
```

Life cycle methods of JSP
JSP contains three life cycle methods.
1) _jspInit()
It is used for instantiation event.
This method will execute just before JES class object creation.
Here JES stands for Java Equivalent Servlet.
2) _jspService()
It is used for request arrival event.
This method will execute when request goes to JSP program.
3) _jspDestroy()
It is used for destruction event.
This method will execute just before JES class object destruction.

Phases in JSP
We have two phases in JSP.
1) Translation phase
In translation phase our JSP program converts to JES class.

2) Request Processing phase
In request processing phase our JES class will be executed and result will send to browser window as dynamic response.

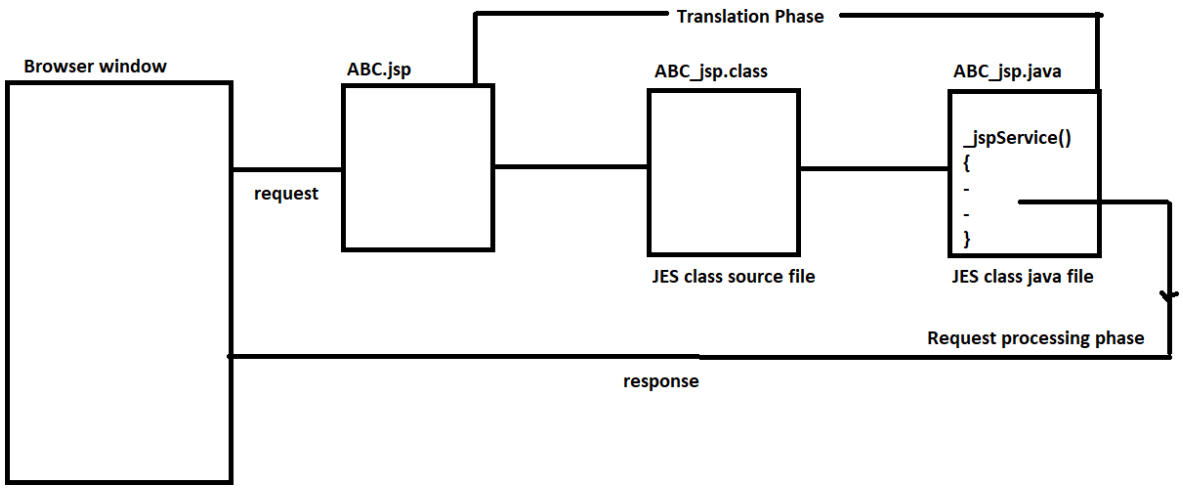

Diagram: jsp2.1
Q) How to enable <load-on-startup> and what happens if we enable <load-on-startup>?
We can enable <load-on-startup> inside web.xml file.

```
web.xml
<web-app>
	<servlet>
		<servlet-name>ABC</servlet-name>
		<jsp-file>/ABC.jsp</jsp-file>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
```

```
                <servlet-name>ABC</servlet-name>
                <url-pattern>/test</url-pattern>
        </servlet-mapping>
</web-app>
```

If we enable <load-on-startup> then translation phase will be perform during server startup or during the deployment of web application.
It means our JES class object will be ready before we give the first request.

JSP Tags/Elements
We have following tags in JSP.
1) Scripting Tags
<u>i) scriptlet tag</u>
ex: <% code %>
<u>ii) expression tag</u>
ex: <%= code %>
<u>iii) declaration tag</u>
ex: <%! code %>
2) Directive Tags
<u>i) page directive tag</u>
ex: <%@page attribute=value %>
<u>ii) include directive tag</u>
ex: <%@include attribute=value %>
3) Standard Tags
ex:
<jsp:include>
<jsp:forward>
<jsp:useBean>
<jsp:setProperty>
<jsp:getProperty>
and etc.
4) Comments
ex: <%-- comment here --%>

i) scriptlet tag
A scriptlet tag is used to declare java code.
syntax: <% code %>

<u>Deployment Directory structure</u>

```
JspApp2
|
|---Java Resources
|
|---Web Content
        |
        |---form.html
        |---process.jsp
        |
        |---WEB-INF
                |
                |---web.xml
```
Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>form.html</u>
```
<form action="process.jsp">
        Name: <input type="text" name="t1"/> <br>
        <input type="submit" value="submit"/>
</form>
```

<u>web.xml</u>
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>process.jsp</u>
```
<center>
        <h1>
                <%
                        String name=request.getParameter("t1");
                        out.println("Welcome :"+name);
                %>
        </h1>
</center>
```

<u>request url</u>

```
        http://localhost:2525/JspApp2/
```

ii) expression tag

The code which is written in expression will return to the output stream of a response so we don't need to write out.println() to print the data.

syntax: `<%=  code   %>`

Note:

Expression tag does not allow semicolon.

Deployment Directory structure

```
JspApp2
|
|---Java Resources
|
|---Web Content
        |
        |---form.html
        |---process.jsp
        |
        |---WEB-INF
                |
                |---web.xml
```

Note:

In above application we need to add "servlet-api.jar" file in project build path.

form.html

```
<form action="process.jsp">
        Name: <input type="text" name="t1"/> <br>
        <input type="submit" value="submit"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>



</web-app>
```

process.jsp

```
<center>
        <h1>
                <%
                        String name=request.getParameter("t1");
                %>
                <%= "Hello :"+name %>
        </h1>
</center>
```

request url

```
        http://localhost:2525/JspApp2/
```

iii) declaration tag

Declaration tag is used to declare fields and methods.

syntax: <%! code   %>

Deployment Directory Structure

```
JspApp3
|
|---Java Resources
|
|---Web Content
|        |
         |---index1.jsp
         |---index2.jsp
         |
         |---WEB-INF
                 |
                 |---web.xml
```

Note:

In above application we need to add "servlet-api.jar" file in project build path.

index1.jsp

```
<center>
<h1>
<%!
                int data=100;
%>
<%=  "The value is ="+data %>
</h1>
</center>
```

index2.jsp

```
<center>
<h1>
<%!
            int cube(int n)
            {
                  return n*n*n;
            }
%>
<%= "Cube of a given number is ="+cube(5) %>
</h1>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

</web-app>
```

request url

```
      http://localhost:2525/JspApp3/index1.jsp
      http://localhost:2525/JspApp3/index2.jsp
```

Exception Handling in JSP

Runtime errors are called exceptions.

Exception may rise any time in our application so handling the exceptions is a safer side for the programmer.

In JSP, There are two ways to handle exceptions.

1) By using errorPage and isErrorPage attributes of page directive tag

2) By using <error-page> element in web.xml file

1) By using errorPage and isErrorPage attributes of page directive tag

Deployment Directory structure

```
JspApp4
|
|--Java Resources
|
|--Web Content
      |
      |----form.html
      |----process.jsp
```

```
          |----error.jsp
          |
          |----WEB-INF
                    |
                    |----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>form.html</u>

```
<form action="process.jsp">
          No1: <input type="text" name="t1"/> <br>
          No2: <input type="text" name="t2"/> <br>
          <input type="submit" value="divide"/>
</form>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

          <welcome-file-list>
                    <welcome-file>form.html</welcome-file>
          </welcome-file-list>

</web-app>
```

process.jsp

```
<%@page errorPage="error.jsp" %>
<%
          String sno1=request.getParameter("t1");
          String sno2=request.getParameter("t2");
          int a=Integer.parseInt(sno1);
          int b=Integer.parseInt(sno2);
          int c=a/b;
%>
<%= "Division of two numbers is ="+c %>
```

<u>error.jsp</u>

```
<%@page isErrorPage="true" %>
<b><i>
          Sorry Exception Occured!! <br>
```

```
	<%= exception %>
</i></b>
```

Request url

```
	http://localhost:2525/JspApp4/
```

2) By using <error-page> element in web.xml file

This approach is recommanded to use because we don't need to defining errorPage element in each jsp file.Defining single entry in web.xml file will handle all types of exceptions.

Deployment Directory structure

```
JspApp4
|
|--Java Resources
|
|--Web Content
	|
	|----form.html
	|----process.jsp
	|----error.jsp
	|
	|----WEB-INF
		|
		|----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

form.html

```
<form action="process.jsp">
	No1: <input type="text" name="t1"/> <br>
	No2: <input type="text" name="t2"/> <br>
	<input type="submit" value="divide"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app				xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

	<error-page>
		<exception-type>java.lang.Exception</exception-type>
		<location>/error.jsp</location>
	</error-page>
	<welcome-file-list>
		<welcome-file>form.html</welcome-file>
	</welcome-file-list>

</web-app>
```

<u>process.jsp</u>

```
<%
	String sno1=request.getParameter("t1");
	String sno2=request.getParameter("t2");
	int a=Integer.parseInt(sno1);
	int b=Integer.parseInt(sno2);
	int c=a/b;
%>
<%= "Division of two numbers is ="+c %>
```

<u>error.jsp</u>

```
<%@page isErrorPage="true" %>
<b><i>
	Sorry Exception Occured!! <br>

	<%= exception %>

</i></b>
```

<u>Request url</u>

```
	http://localhost:2525/JspApp4/
```

JSP to Database Communication

<u>Deployment Directory structure</u>

```
JspApp5
|
|---Java Resources
|
|---Web Content
	|
	|---form.html
	|---process.jsp
	|
	|---WEB-INF
		|
		|---web.xml
		|
                    |------lib
                              |
                              |----ojdbc14.jar
```
Note:
In above application we need to add "servlet-api.jar" and "ojdbc14.jar" file in project build path.
Copy and paste "ojdbc14.jar" file in project build path.

<u>form.html</u>
```
<form action="process.jsp">
        No: <input type="text" name="t1"/> <br>
        Name: <input type="text" name="t2"/> <br>
        Address: <input type="text" name="t3"/> <br>
        <input type="submit" value="submit"/>
</form>
```

<u>web.xml</u>
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>process.jsp</u>
```
<%@page import="java.sql.*" buffer="8kb"  language="java" %>
<%
        String sno=request.getParameter("t1");
        int no=Integer.parseInt(sno);

        String name=request.getParameter("t2");

        String add=request.getParameter("t3");

        //insert the data into database
        Connection con=null;
        PreparedStatement ps=null;
        String qry=null;
        int result=0;
        try
        {
                Class.forName("oracle.jdbc.driver.OracleDriver");

        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system",
"admin");
                qry="insert into student values(?,?,?)";
                ps=con.prepareStatement(qry);
                //set the values
                ps.setInt(1,no);
                ps.setString(2,name);
                ps.setString(3,add);

                //execute
                result=ps.executeUpdate();

                if(result==0)
                        out.println("No Record inserted");
                else
                        out.println("Record inserted");

                ps.close();
                con.close();
        }
        catch(Exception e)
        {
                out.println(e);
        }
%>
```

Request url

```
        http://localhost:2525/JspApp5/
```

Actions Tags

Action tags are used to perform particular task.
Action tags are used to control the web pages and uses java beans.
Action tags are executed dynamically at runtime.
Action tags contain standard tags but we don't have xml tags.

Action tags are divided into two types.

1) Standard Action tags
2) Custom Action tags

1) Standard Action tags
Built-In action tags are called standard action tags.
We have following list of standard action tags.

ex:
```
	<jsp:include>
	<jsp:forward>
	<jsp:useBean>
	<jsp:setProperty>
	<jsp:getProperty>
	and etc.
```

Action forward tag
In action forward the output of source jsp program will be discarded and output of destination jsp program goes to browser window as dynamic response.
It internally uses servlet API functionality called rd.forward(req,res).
syntax:		<jsp:forward page="page_name"/>

Deployment Directory structure
```
JspApp6
|
|---Java Resources
|
|---Web Content
	|
	|--A.jsp
	|--B.jsp
	|
	|---WEB-INF
		|
		|--web.xml
```
Note:
In above application we need to add "servlet-api.jar" file in project build path.

A.jsp
```
<b><i> Begining of A.jsp</i></b>
<br>
<jsp:forward page="B.jsp"/>
<br>
<b><i>Ending of A.jsp</i></b>
```

B.jsp
```
<b><i>This is B.jsp</i></b>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>A.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

Request url

```
        http://localhost:2525/JspApp6/
```

Action include

In action include the output of source jsp program and destination jsp program combinely goes to browser window as dynamic response.

It internally uses servlet API functionality called rd.include(req,res);

syntax:

```
        <jsp:include page="page_name"/>
```

Deployment Directory structure

```
JspApp6
|
|---Java Resources
|
|---Web Content
        |
        |--A.jsp
        |--B.jsp
        |
        |---WEB-INF
                |
                |--web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

A.jsp

```
<b><i> Begining of A.jsp</i></b>
<br>
```

```
<jsp:include page="B.jsp"/>
<br>
<b><i>Ending of A.jsp</i></b>
```

B.jsp
```
<b><i>This is B.jsp</i></b>
```

web.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>A.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

Request url

  http://localhost:2525/JspApp6/

JSP to Java Bean Communication

JSP to java bean communication is possible by using three tags.

1) <jsp:useBean> tag

   It is used to locate and create bean class object.

2) <jsp:setProperty> tag

   It is used to set the value to bean object and calls setter methods.

3) <jsp:getProperty> tag

   It is used to get the value from bean object and calls getter methods.

Note:

  All above tags are independent tags.

ex:1

Deployment Directory structure
```
JspApp7
|
|--Java Resources
        |
        |------src
                |
                |---com.ihub.www
```

```
                              |
                              |---CubeNumber.java
|---Web Content
          |
          |-----process.jsp
          |
          |-----WEB-INF
                    |
                    |----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>CubeNumber.java</u>

```java
package com.ihub.www;

public class CubeNumber
{
	public int cube(int n)
	{
		return n*n*n;
	}
}
```

<u>process.jsp</u>

```jsp
<jsp:useBean id="cn" class="com.ihub.www.CubeNumber"></jsp:useBean>
<%=  "Cube of a given number is ="+cn.cube(5)  %>
```

<u>web.xml</u>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
	<welcome-file>process.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>Request url</u>

	http://localhost:2525/JspApp7/

ex:2

<u>Deployment Directory structure</u>

```
JspApp8
|
|---Java Resources
          |
          |-----src
                    |
                    |----com.ihub.www
                              |
                              |---User.java
|---Web Content
          |
          |-----form.html
          |-----process.jsp
          |
          |-----WEB-INF
                    |
                    |-----web.xml
```

Note:
In above application we need to add "servlet-api.jar" in project build path.

form.html

```
<form action="process.jsp">

        UserName: <input type="text" name="username"/> <br>
        Password: <input type="password" name="password"/> <br>
        Email: <input type="text" name="email"/> <br>
        <input type="submit" value="submit"/>

</form>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>
```

```
</web-app>
```

User.java

```
package com.ihub.www;

public class User
{
	private String username;
	private String password;
	private String email;

	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}

}
```

process.jsp

```
<jsp:useBean id="u" class="com.ihub.www.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>
Records are : <br>
<jsp:getProperty property="username" name="u"/> <br>
<jsp:getProperty property="password" name="u"/> <br>
<jsp:getProperty property="email" name="u"/> <br>
```

Request url

```
	http://localhost:2525/JspApp8/
```

2) Custom tags in JSP

To create a custom tag in JSP we need to use taglib directory.

We can declare taglib directory as follow.

ex: <%@taglib uri="uriofthetaglibrary"   prefix="prefixoftaglibrary" %>

Deployment Directory structure

```
JspApp9
|
|----Java Resources
|         |
          |------src
                   |
                   |--com.ihub.www
                            |
                            |----CubeNumber.java
|
|----Web Content
          |
          |------process.jsp
          |
          |------WEB-INF
                   |
                   |-----web.xml
                   |-----mytags.tld
                   |
                   |------lib
                            |
                            |---jsp-api.jar
```

Note:

In above application we need to add "servlet-api.jar" and "jsp-api.jar" file in project build path.

Copy and paste "jsp-api.jar" file inside "WEB-INF/lib" folder sperately.

process.jsp

```
<%@taglib uri="/WEB-INF/mytags.tld" prefix="ihub" %>
Cube of a given number is : <ihub:cube  number="5"/>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <welcome-file-list>
```

```
            <welcome-file>process.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>mytags.tld</u>

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-taglib</uri>

  <tag>
            <name>cube</name>
            <tag-class>com.ihub.www.CubeNumber</tag-class>
            <attribute>
                        <name>number</name>
                        <required>true</required>
            </attribute>
  </tag>



</taglib>
```

<u>CubeNumber.java</u>

```
package com.ihub.www;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class CubeNumber extends TagSupport
{
            private int number;

            //setter method
            public void setNumber(int number)
```

```
        {
                this.number=number;
        }

        public int doStartTag()throws JspException
        {
                try
                {
                        JspWriter out=pageContext.getOut();
                        out.println(number*number*number);
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }

                return SKIP_BODY;
        }
}
```

Request url

http://localhost:2525/JspApp9/

MVC in JSP

MVC stands for Model View Controller.
It is one of the design pattern which seperates business logic , persistence logic and data.
Controller is an interface between Model and View.
Controller will intercept all incoming request.
Model contains data and sometimes it contains business logic.
View contains presentation i.e UI.

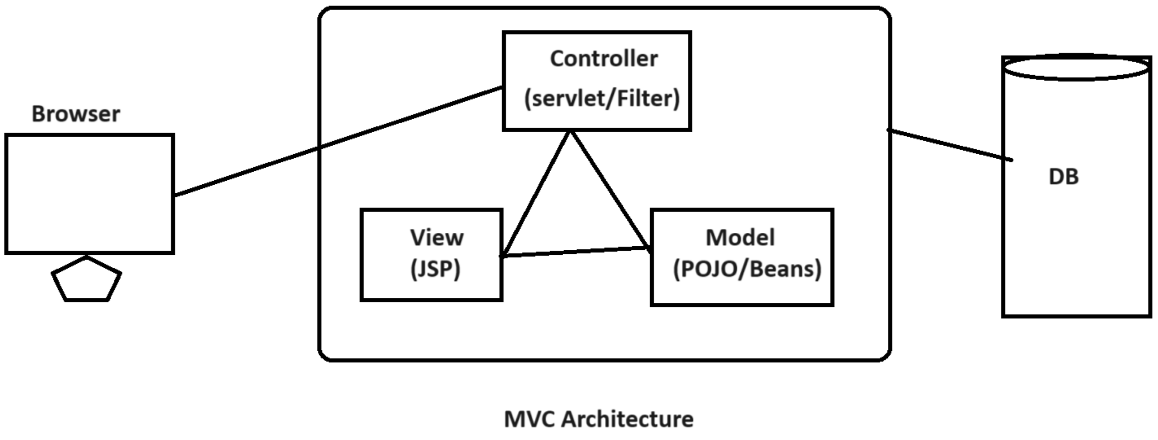

Diagram: jsp5.1

Deployment Directory structure

```
MVCApp
|
|----Java Resources
|        |
         |------src
                  |
                  |--com.ihub.www
                           |
                           |----LoginBean.java
                           |----LoginSrv.java
|
|----Web Content
         |
         |------form.html
         |------view.jsp
         |------error.jsp
         |
         |------WEB-INF
                  |
                  |-----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

form.html

```
<form action="test">
        UserName: <input type="text" name="username"/> <br>
        Password: <input type="password" name="password"/> <br>
        <input type="submit" value="login"/>
</form>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <servlet>
        <servlet-name>LoginSrv</servlet-name>
        <servlet-class>com.ihub.www.LoginSrv</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>LoginSrv</servlet-name>
        <url-pattern>/test</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
        <welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

<u>LoginBean.java</u>

```
package com.ihub.www;

public class LoginBean
{
        private String username;
        private String password;

        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }

}
```

<u>LoginSrv.java</u>

```
package com.ihub.www;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class LoginSrv extends HttpServlet
{
	protected void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{
		PrintWriter pw=res.getWriter();
		res.setContentType("text/html");

		//reading form data
		String uname=req.getParameter("username");
		String pass=req.getParameter("password");

		//create login bean object and set the values
		LoginBean lb=new LoginBean();
		lb.setUsername(uname);
		lb.setPassword(pass);

		req.setAttribute("bean",lb);

		if(pass.equals("admin"))
		{
			RequestDispatcher rd=req.getRequestDispatcher("view.jsp");
			rd.forward(req, res);
		}
		else
		{
			RequestDispatcher rd=req.getRequestDispatcher("error.jsp");
			rd.forward(req, res);
		}

		pw.close();
	}
}
```

<u>view.jsp</u>

```
<%@page import="com.ihub.www.LoginBean" %>
<%
	LoginBean lb=(LoginBean)request.getAttribute("bean");
%>
Details are :  <br>

<%= lb.getUsername() %> <br>
<%= lb.getPassword() %> <br>
```

error.jsp

```
<b><i>
	<font color="red">Sorry! incorrect username or password</font>
</i></b>
<%@include file="form.html" %>
```

request url

```
	http://localhost:2525/MVCApp/
```

Implicit objects in JSP

Object which can be used directly without any configuration is called implicit object.
Implicit object is created by the web container which is available for every JSP page.
JSP contains 9 implicit objects as shown below.
ex:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	pageContext
page	Object
exception	Throwable

response object

In jsp, response is a implicit object of type HttpServletResponse.
It can be used to add or manipulate response such as redirect response or another resources,send error and etc.

Deployment Directory

```
JspApp10
|
|---Java Resources
|
|---Web Content
	|
	|---index.html
	|
	|---process.jsp
	|
	|------WEB-INF
		|
		|----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

form.html
```
<center>
	<h1>
		<a href="process.jsp"> Facebook Page </a>
	</h1>
</center>
```

process.jsp

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

web.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

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

request url
----
	http://localhost:2525/JspApp10/

config object
It is an implicit object of type ServletConfig.
The config object is created by web container for each jsp page.
This object is used to read initialized parameters for a perticular jsp page.

Deployment Directory
```
JspApp11
|
|---Java Resources
|
|---Web Content
```

```
          |
          |---index.html
          |
          |---process.jsp
          |
          |------WEB-INF
                    |
                    |----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

<u>index.html</u>

```
<center>
          <h1>
                    <a href="test"> clickMe </a>
          </h1>
</center>
```

<u>process.jsp</u>

```
<%
          String value=config.getInitParameter("driver");
%>
<%= value  %>
```

<u>web.xml</u>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
          <servlet>
                    <servlet-name>ABC</servlet-name>
                    <jsp-file>/process.jsp</jsp-file>
                    <init-param>
                              <param-name>driver</param-name>
                              <param-value>oracle.jdbc.driver.OracleDriver</param-value>
                    </init-param>
          </servlet>
          <servlet-mapping>
                    <servlet-name>ABC</servlet-name>
                    <url-pattern>/test</url-pattern>
          </servlet-mapping>
          <welcome-file-list>
```

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

request url

```
	http://localhost:2525/JspApp11/
```

application object

In jsp, application is an implicit object of type ServletContext.
This instance of ServletContext is created only once by the web container.
This object is used to read initialized parameters from configuration file web.xml file.

Deployment Directory structure

```
JspApp12
|
|---Java Resources
|
|---Web Content
	|
	|---index.html
	|
	|---process.jsp
	|
	|------WEB-INF
		|
		|----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

index.html

```
<center>
	<h1>
		<a href="test"> clickMe </a>
	</h1>
</center>
```

process.jsp

```
<%
	String value=application.getInitParameter("driver");
%>
<center>
	<%= value  %>
</center>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
	<servlet>
		<servlet-name>ABC</servlet-name>
		<jsp-file>/process.jsp</jsp-file>
	</servlet>
	<servlet-mapping>
		<servlet-name>ABC</servlet-name>
		<url-pattern>/test</url-pattern>
	</servlet-mapping>
	<context-param>
			<param-name>driver</param-name>
			<param-value>oracle.jdbc.driver.OracleDriver</param-value>
	</context-param>
	<welcome-file-list>
		<welcome-file>index.html</welcome-file>
	</welcome-file-list>
</web-app>
```

request url

```
	http://localhost:2525/JspApp12/
```

sessionobject

In JSP, session is an implicit object of type HttpSession.
It is used to get or set the session formation.

Deployment Directory structure

```
JspApp13
|
|---Java Resources
|
|---Web Content
	|
	|---form.html
	|
	|---first.jsp
	|
	|---second.jsp
	|
	|------WEB-INF
```

```
                    |
                    |----web.xml
```

Note:
In above application we need to add "servlet-api.jar" file in project build path.

form.html
```
<form action="first.jsp">
	Name: <input type="text" name="t1"/> <br>
	<input type="submit" value="submit"/>
</form>
```

first.jsp
```
<%
	String name=request.getParameter("t1");
	out.println("Hello :"+name);
	session.setAttribute("pname", name);
%>
<center>
	<a href="second.jsp"> click to move for second.jsp</a>
</center>
```

second.jsp
```
<%
	String name=(String)session.getAttribute("pname");
	out.println("Welcome :"+name);
%>
```

web.xml
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
	<welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Request url
```
	http://localhost:2525/JspApp13/
```

pageContext object

In jsp, pageContext is an implicit object of type pageContext class.

The pageContext object can be used to set, get, remove attributes from one the following scopes., page, request, session, application

In JSP, page scope is a default scope.

Deployment Directory structure

```
JspApp14
|
|---Java Resources
|
|---Web Content
        |
        |---form.html
        |
        |---first.jsp
        |
        |---second.jsp
        |
        |------WEB-INF
                |
                |----web.xml
```

Note:

In above application we need to add "servlet-api.jar" file in project build path.

form.html

```
<form action="first.jsp">
        Name: <input type="text" name="t1"/> <br>
        <input type="submit" value="submit"/>
</form>
```

first.jsp

```
<%
        String name=request.getParameter("t1");
        out.println("Hello :"+name);
        pageContext.setAttribute("pname", name,pageContext.SESSION_SCOPE);
%>
<center>
        <a href="second.jsp"> click to move for second.jsp</a>
</center>
```

second.jsp

```
<%
	String
name=(String)pageContext.getAttribute("pname",pageContext.SESSION_SCOPE);
	out.println("Welcome  Bro :"+name);
%>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <welcome-file-list>
	<welcome-file>form.html</welcome-file>
  </welcome-file-list>

</web-app>
```

request url

```
	http://localhost:2525/JspApp14
```

Junit

Junit is a testing framework which is used to perform unit testing.
Unit testing is a process to test small piece of code working as per the requirement or not.
As a part of TDD(Test Driven Development) , It is highly recommanded to perform unit testing by the web developers.
To perform unit testing we need to create test cases or test suit.

Steps to perform unit testing

step1:
	Launch eclipse IDE by choosing workspace location.
step2:
	Create a java project i.e JunitProj.
	ex:
		File --> new --> project --> java project --> Next -->
		project Name : JunitProj --> Next --> Finish.
step3:
	Create "com.ihub.www" package inside "src" folder.
	ex:
		Right click to src folder --> new --> package -->
		Name : com.ihub.www --> finish.
step4:
	Create "DemoApp.java" file inside "com.ihub.www" package.

ex:
    right click to com.ihub.www package --> new --> class -->
    Name: DemoApp --> Finish.

DemoApp.java

```
package com.ihub.www;
public class DemoApp
{
        public String concatination(String str1,String str2)
        {
                return str1+str2;
        }

        public int sum(int a,int b)
        {
                return a+b;
        }
}
```

step5:

Create a Junit test cases for java methods.

ex:
    right click to DemoApp.java file --> new --> others -->
    Junit --> Junit Test case --> Next --> select the methods
    to whome we want to apply the test cases --> Finish.

step6:

Write the logic to test the test cases inside "DemoAppTest.java" file.

ex:

DemoAppTest.java

```
package com.ihub.www;
import junit.framework.TestCase;
public class DemoAppTest extends TestCase {
        public void testConcatination() {
                DemoApp da=new DemoApp();
                String result=da.concatination("ihub", "talent");
                assertEquals("ihubtalent",result );
        }
        public void testSum() {
                DemoApp da=new DemoApp();
                int result=da.sum(10, 20);
                assertEquals(30, result);
        }

}
```

step7:

  Run the Junit test cases.
  ex:
    Right click to the DemoAppTest.java --> run as -->
    Junit test case.

Note:

  Green color indicates test case is passed.
  Brown color indicates test case is failed.

Maven

Maven is a project building management tool.
Maven contains POM.xml file.
Here POM stands for Project Object Model.
A pom.xml file contains dependencies , goals , descriptors and etc.

<u>Steps to develop Maven project</u>

step1:

  Launch eclipse IDE by choosing workspace location.

step2:

  create a dynamic web project.
  ex:
    File --> new --> dynamic project --> Name : MavenProj
    ---> Next --> Next --> generate web.xml file --> finish.

step3:

  Convert dynamic web project to Maven project.
  ex:
    Right click to dynamic project --> configure -->
    convert to maven project -->
    Group ID : com.ihub.www
    Artifact ID : MavenProj
    Name : MavenProj
    Description: Demostration on Maven project --> finish.

step4:

  Create a "ABC.jsp" file inside "Web Content" folder.
  ex:
  <u>ABC.jsp</u>

```
<center>
	<h1>
		This is Maven Project Demo
	</h1>
</center>
```

step5:

  Add "servlet-api.jar" file manven depedency inside pom.xml file.
  ex:

```
pom.xml
-
-
 <dependencies>
          <dependency>
                    <groupId>javax.servlet</groupId>
                    <artifactId>servlet-api</artifactId>
                    <version>2.5</version>
                    <scope>provided</scope>
          </dependency>
</dependencies>
<build>
--
--
--
```

step6:

Run the maven project.

ex:

Right click to MavenProject --> run as --> run on server.

step7:

Test the application by using below request url.

ex:

http://localhost:2525/MavenProj/ABC.jsp

How to convert Maven project or Dynamic project to war file

step1:

Make sure Dynamic or Maven project is ready in eclipse IDE.

step2:

convert Dynamic or Maven project to war file .

ex:

Right click to MavenProj --> export --> war file -->
Destination : Desktop(choose) --> open --> finish.

GIT/GITHUB

Q) Difference between GIT vs GITHUB ?

GIT	GITHUB
It is distributed version control system to track changes of each file in a project.	It is a web-based hosting service for git.
It contains local repository.	It contains remote repository.
It is command line tool.	It is GUI.
It is locally installed.	It is hosted on web.

Q) Types of stages of Git?

We have three stages in git.

Working Directory:

the file exists, but is not part of git's version control.

staging area:

the file has been added to git's version control but changes have not been committed

Repository:

the change has been committed

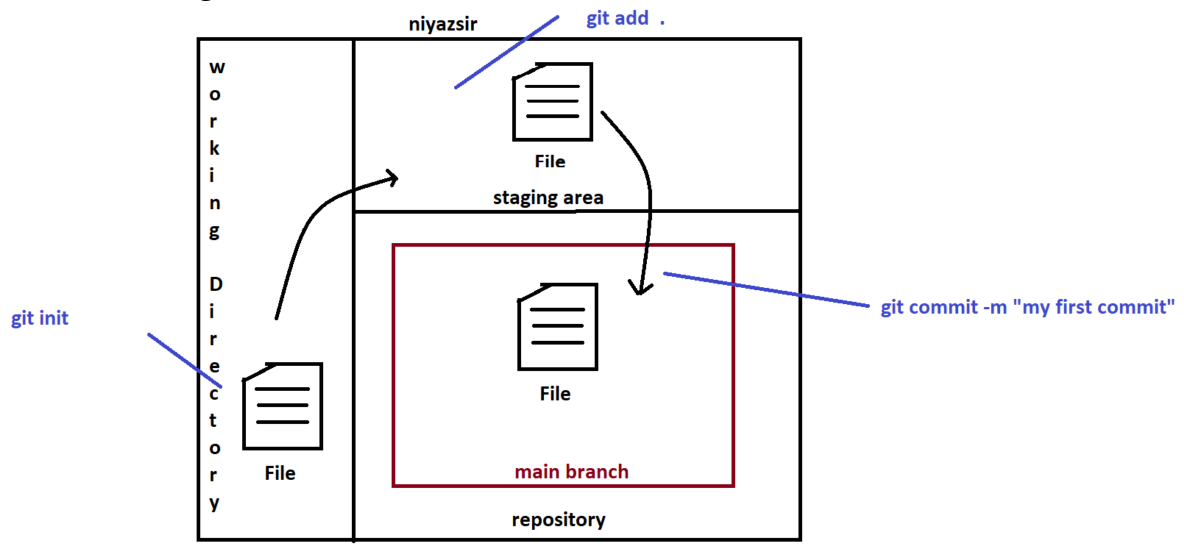

Diagram: git

Remote repository github

https://github.com/NiyazulHasan/practice

Q) Write a git command to initialized empty repository?

git init

Q) Write a git command to check the status?

git status

Q) Write a git command to check the branch?

git branch

Q) Write a git command to add remote repository?

git remote add origin https://github.com/NiyazulHasan/practice

Q) Write a git command to check the remote repository

git remote -v

Q) Write a git command to commit the changes?

git commit -m "comment here"

Q) Write a git command to push the code to remote origin?

```
git push -f origin main
```

Q)Write a git command to clone the project?

```
git clone <url>
```

Q)Write a git command to pull request?

```
git pull <url>
```

Q)Write a git command to move from master branch to main branch?

```
git branch --move master main
```

SPRING BOOT

Limitations with Spring Framework

In spring framework a programmer is responsible for following things.

Adding the depencies or jar files

Performing the configurations (applicationContext.xml)

Managing the physical servers like Tomcat and etc.

Arranging the physical database like Oracle,MySQL and etc.

```
     Developer
         |
    Spring Boot
         |
  Spring Framework
```

Spring Boot

It is an open source java based application framework developed by Pivotal Team.

It provides RAD (Rapid Application Development) features to spring based applications.

It is a production ready grade spring applications with minimum configurations.

In short , **spring boot is a combination of**

Spring Framework + Embedded database + Embedded server

Spring boot does not support xml configuration instead we will use annotations.

Advantages of Spring Boot

It is used to create standalone application which can be started by using java -jar.

It provide production ready grade features like metrix, healthcheck, externalized configurations and etc.

It provides optionate starters to simplify the Maven configurations.

It does not support xml configurations.

It test the web application by using HTTP servers like Tomcat, Jetty or Undertow.

It provides CLI tool for testing and developing spring boot applications.

Interview Questions

Q) What is Spring Boot?

It is an open source java based application framework developed by pivotal team.It provides RAD features to spring based applications.It is a production ready grade spring applications with miniconfigurations.

Q) How many components are there in Spring Boot?

We have four components in spring boot.

1) AutoConfiguration
2) Starters
3) CLI tool
4) Actuators

Q) Where we will do configurations in spring boot?

There are two ways to do configurations in spring boot.

1) application.properties (default)
2) application.yml

Q) List out some embedded servers present in spring boot?

We have following list of embedded servers present in spring boot.

1) Tomcat
2) Jetty
3) Undertow

Q) List out some embedded databases present in spring boot?

We have following list of embedded databases present in spring boot.

1) H2
2) HSQL
3) Derby

Q) In how many ways we can create spring boot application?

There are two ways to create spring boot applications.

1) Using Spring Initializr
2) Using STS or IntelliJ IDE.

Q) Explain @SpringBootApplication Annotation?

This annotation is a combination of three annotations.

1) @EnableAutoConfiguration :
   It enables Spring Boot's auto-configuration mechanism.
2) @ComponentScan :
   It scans on the package where the application is located.
3) @Configuration :
   It allows us to register extra beans in the context.

Spring Initializr
It is a web based tool which is used to generate spring boot project structure.
ex: https://start.spring.io/

<u>Steps to develop first spring boot application</u>
step1:
        Goto spring initializr
        ex:
                https://start.spring.io/
step2:
        Create a spring boot project i.e FirstSB.
        ex:

```
project      : Maven
Language     : Java
Dependencies : (don't add)
 Spring Boot  : 3.1.1

Project Metadata
 Group     :    com.ihub.www
Artifact   :    SBApp1
Name       :    SBApp1
Description :   Demostration on Spring Boot
Package name:   com.ihub.www
packaging  :   jar
Java       :   8
```

        ---> click on Generate button.
step3:
        Download and Install STS IDE.
        STS is a eclipse Based Environment.
        ex: https://spring.io/tools
step4:
        Launch STS IDE by choosing workspace location.
step5:
        Extract "SBApp1.zip" file in any loctation.
step6:
        Open "SBApp1" project in STS IDE.
        ex:
                File --> Open project from file system --> click to directory button.
                --> select FirstSB folder --> Finish.
step7:
        Add custom message in SApp1Application.java file.
step8:
        Run the spring boot appilication.
        ex: right click to the project --> run as --> spring boot application.

Spring Boot Starters
Spring Boot provides a number of starters that allow us to add jars in the classpath.
Spring Boot built-in starters make development easier and rapid.
Spring Boot Starters are the dependency descriptors.
In the Spring Boot Framework, all the starters follow a similar naming pattern:
    spring-boot-starter-*, where * denotes a particular type of application.
ex:
    spring-boot-starter-test
    spring-boot-starter-web
    spring-boot-starter-validation (bean validation)
    spring-boot-starter-security
    spring-boot-starter-data-jpa
    spring-boot-starter-data-mongodb
    spring-boot-starter-mail

Third-Party Starters
We can also include third party starters in our project.
The third-party starter starts with the name of the project.
ex:
    abc-spring-boot-starter.

Spring Boot Starter Web
There are two important features of spring-boot-starter-web.
    It is compatible for web development
    AutoConfiguration
If we want to develop a web application,we need to add the following dependency in pom.xml file.
ex:
```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <version>2.2.2.RELEASE</version>
</dependency>
```
Spring web starter uses Spring MVC, REST and Tomcat as a default embedded server.
The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development.
By default, the spring-boot-starter-web contains the following tomcat server dependency:
ex:
```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <version>2.0.0.RELEASE</version>
        <scope>compile</scope>
</dependency>
```

The spring-boot-starter-web ,auto-configures the following things that are required for the web development:

1) Dispatcher Servlet
2) Error Page
3) Web JARs for managing the static dependencies
4) Embedded servlet container

Spring Boot + JSP Application

Project structure

```
SBApp2
|
|----src/main/java
|         |
|         |----com.ihub.www (base package)
|                   |
|                   |--SBApp2Application.java
|                   |--HomeController.java
|---src/main/resources
|         |
|         |-----application.properties
|
|---src/test/java
|         |
|         |-----SpringBootApp3ApplicationTests.java
|
| --
| --
| --
|-----src
          |
          |----main
                    |
                    |----webapp
                              |
                              |----pages
                              |         |
                                        |----index.jsp
|---pom.xml
|
|
```

step1:

Create a spring starter project.

ex:

```
        File --> new --> spring starter project -->
                        Name : SpringBootApp3
                        Group: com.ihub.www
                        Artifact: SpringBootApp3
                        Description: This is Spring Boot Application with JSP
                        package : com.ihub.www  ---> next -->
                        Starter: Spring Web --> next --> Finish.
```

step2:

create a HomeController class inside "src/main/java".

ex:

Right click to package(com.ihub.www) --> new --> class -->
Class: HomeController -->finish.

step3:

Add @Controller annotation and "@RequestMapping" annotation
inside HomeController class.

<u>HomeController.java</u>

```
package com.ihub.www;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

    @RequestMapping("/home")
    public String home()
    {
        return "index";
    }
}
```

step4:

create a "webapp" and "pages" folder inside "src/main" folder for adding JSP files.

ex:

```
        |-----src
              |
              |----main
                   |
                   |----webapp
                        |
                        |-----pages
```

step5:
    create "index.jsp" file inside "src/main/webapp/pages/" folder.
    ex:
        Right click to pages folder--> new --> file --->
        File Name: index.jsp --> finish.

<u>index.jsp</u>

```
<center>
        <h1>
                I love Spring Boot Programming
        </h1>
</center>
```

step6:
    Add "Tomcat Embed Jasper" dependency to read the jsp file.
    ex:

```
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Note:
    Embedded Tomcat server does not have Jasper. So we need to add above dependency.

step7:
    Configure tomcat server port number and jsp file.

<u>application.properties</u>

```
server.port=9090

spring.mvc.view.prefix=/pages/
spring.mvc.view.suffix=.jsp
```

step8:
    Run Spring Boot application.
    ex:
        Right click to MVCApp2 --> run as --> spring boot application.

step9:
    Test the application with below request url.
    ex:
        http://localhost:9090/home

Note:-
If you are not getting output please add tomcat dependency in pom.xml file seperately.
ex:

```
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
        <version>3.1.1</version>
    </dependency>
```

Interview question

Q) To create a spring mvc based web application we need to add which starter?

spring-boot-stater-web

Q) In spring boot mvc based web application who will pass HTTP request to controller?

DispatcherServlet

Q) Tomcat embedded server by default runs under which port no?

8080

Q) How to change tomcat embedded server port no?

application.properties:-   server.port = 9090

Spring Data JPA

Spring Data JPA handles most of the complexity of JDBC-based database access and ORM (Object Relational Mapping).
It reduces the boilerplate code required by JPA(Java Persistence API).
It makes the implementation of your persistence layer easier and faster.
Spring Data JPA aims to improve the implementation of data access layers by reducing the effort to the amount that is needed.
Spring Boot provides spring-boot-starter-data-jpa dependency to connect Spring application with relational database efficiently.
ex:

```
    <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.2.2.RELEASE</version>
    </dependency>
```

The spring-boot-starter-data-jpa internally uses the spring-boot-jpa dependency.
Spring Data JPA provides three repositories are as follows:

CrudRepository:

It offers standard create, read, update, and delete It contains method like findOne(), findAll(), save(), delete(), etc.

PagingAndSortingRepository:

It extends the CrudRepository and adds the findAll methods. It allows us to sort and retrieve the data in a paginated way.

JpaRepository:
It is a JPA specific repository It is defined in Spring Data Jpa. It extends the both repository CrudRepository and PagingAndSortingRepository. It adds the JPA-specific methods, like flush() to trigger a flush on the persistence context.

Spring Boot application to interact with H2 Database

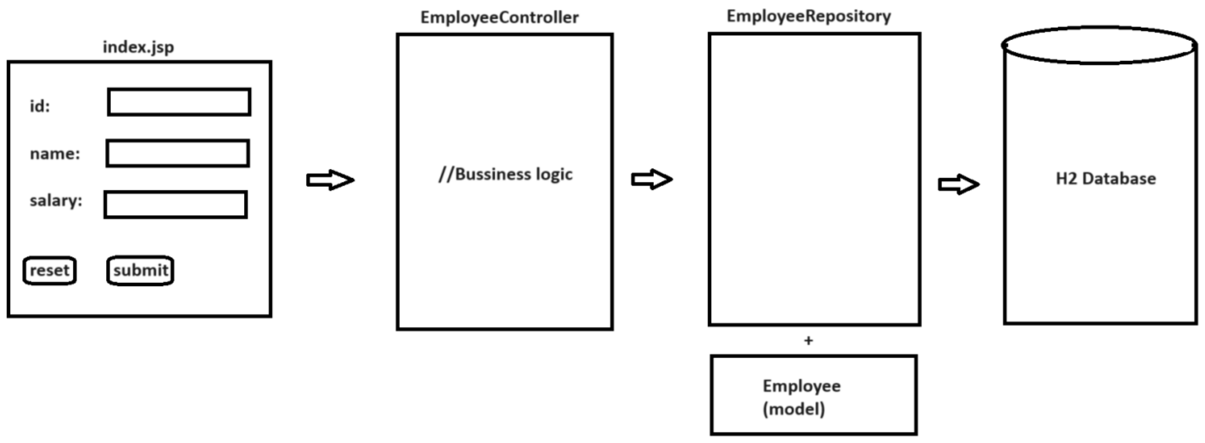

Diagram: sb3.1

project structure

```
SBApp3
|
|----src/main/java
          |
          |---com.ihub.www
          |         |
                    |---SBApp3Application.java
          |
          |---com.ihub.www.controller
                    |
                    |---EmployeeController.java (Class)
          |
          |---com.ihub.www.repository
                    |
                    |---EmployeeRepository.java (Interface)
          |
          |---com.ihub.www.model
                    |
                    |---Employee.java (Class)
|
|
|----src/main/resources
          |
          |---application.properties
          |
|-----src
        |
        |-----main
                |
                |---webapp
                        |
                        |----index.jsp
```
step1:

        Create a spring boot starter project i.e MVCApp2.
        ex:
                starters:
                                spring web
                                spring data jpa
                                H2 Database

step2:

        Add "Tomcat Embed Jasper" dependency to read the jsp file inside pom.xml.
        ex:

```
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```
step3:

        Create a EmployeeController inside "com.ihub.www.controller" package.

EmployeeController.java

```
package com.ihub.www.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import com.ihub.www.model.Employee;
import com.ihub.www.repository.EmployeeRepository;

@Controller
public class EmployeeController
{
        @Autowired
        EmployeeRepository employeeRepository;

        @RequestMapping("/")
        public String home()
        {
                return "index.jsp";
        }
```

```
	@RequestMapping("/addEmp")
	public String addEmployeeDetails(Employee e)
	{
		employeeRepository.save(e);

		return "index.jsp";
	}
}
```
step4:

Create index.js file inside "src/main/webapp" folder.

<u>index.js</u>
```
<form action="addEmp">
	<table align="center">
		<caption>Enter the Details</caption>
		<tr>
			<td>Employee Id </td>
			<td><input type="text" name="empId"/></td>
		</tr>
		<tr>
			<td>Employee Name </td>
			<td><input type="text" name="empName"/></td>
		</tr>
		<tr>
			<td>Employee Salary </td>
			<td><input type="text" name="empSal"/></td>
		</tr>
		<tr>
			<td><input type="reset" value="reset"/></td>
			<td><input type="submit" value="submit"/></td>
		</tr>
	</table>
</form>
```
step5:

Create a Employee.java file inside "com.ihub.www.model" package.

<u>Employee.java</u>
```
package com.ihub.www.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table
```

```
public class Employee
{
	@Id
	private int empId;
	@Column
	private String empName;
	@Column
	private double empSal;

	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
}
```

step6:

Create a EmployeeRepository.java interface inside "com.ihub.www.repository" package.

<u>EmployeeRepository.java</u>

```
package com.ihub.www.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.ihub.www.model.Employee;

@Repository
public interface EmployeeRepository extends CrudRepository<Employee,Integer>
{

}
```

step7:
Configure server port and h2 database properties inside application.properties file.

application.properties

```
server.port=9090

spring.datasource.url= jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

step8: Run the spring boot starter project.

step9: Test the application by using below request url.
ex: http://localhost:9090
http://localhost:9090/h2-console

RestController:

RestController is used for making restful web services with the help of the @RestController annotation.

This annotation is used at the class level and allows the class to handle the requests made by the client.

The main difference between the @RestController and the @Controller is that the @Restcontroller is a combination of the @controller and @ResponseBody annotation.

We have following HTTP methods along with rest annotations.

HTTP Methods	Annotations
GET	@GetMapping
POST	@PostMapping
PUT	@PutMapping
Delete	@DeleteMapping
and etc.	

Spring Boot Application using @RestController

Project structure

```
RestApp
|
|----src/main/java
|        |
|        |----com.ihub.www
|                 |
|                 |--RestAppApplication.java
|                    |--HomeController.java
|---src/main/resources
|         |
|         |-----application.properties
|
|---src/test/java
|         |
|         |-----RestAppApplicationTests.java
|
| --
| --
| --

|---pom.xml
|
```

step1: Create a spring starter project.
ex:
File --> new --> spring starter project -->
Name : RestApp
Group: com.ihub.www
Artifact: RestApp
Description: This is Spring Boot Application
package : com.ihub.www ---> next -->
Starter: Spring Web --> next --> Finish.

step2: create a HomeController class inside "src/main/java".
ex:
Right click to package(com.ihub.www) --> new -->
class --> Class: HomeController -->finish.

step3: Add @Controller annotation and "@RequestMapping" annotation inside HomeController class.

<u>HomeController.java</u>

```
package com.ihub.www;

import org.springframework.stereotype.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HomeController {

        @GetMapping("/")
        public String home()
        {
                return "Rest Controll Example";
        }
}
```
step4: create "index.jsp" file inside "src/main/webapp/pages/" folder.

ex:

Right click to pages folder--> new --> file --->
File Name: index.jsp --> finish.

step5: Configure tomcat server port number and jsp file.

application.properties

```
server.port=9191
```

step6: Run Spring Boot application.

ex:

Right click to RestApp --> run as --> spring boot application.

step7: Test the application with below request url.

ex:

http://localhost:9191/

Q) Difference between Monolethic Architecture vs Microservice Architecture?

Monolithic Architecture

Monolith means composed all in one piece.

The Monolithic application describes a single-tiered software application in which different components combined into a single program from a single platform.

In Monolithic Architecture we are developing every service individually and at end of the development we are packaging all services as single war file and deploying in a server.

Let’s take an example of E-commerce website where we have basic and common option of Customer

Service, Product Service and Cart Service which a customer can access through browser. When we launch the application It is deployed as a single monolithic application.It means we will have only one single instance.

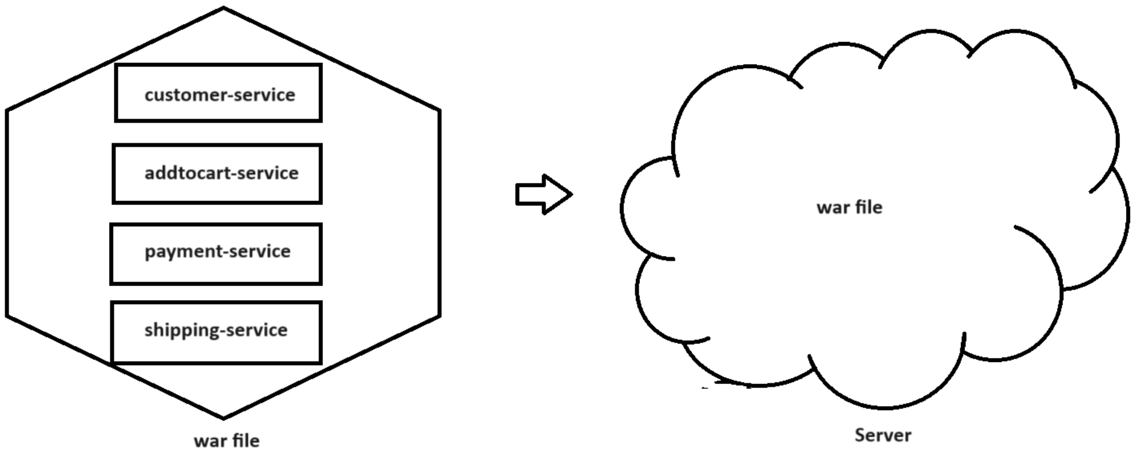

Diagram: sb4.1

Advantages

1) Simple to develop

   At the beginning of a project it is much easier to go with Monolithic Architecture.

2) Simple to test

   We can implement end-to-end testing by simply launching the application and testing the UI with Selenium.

3) Simple to deploy

   we have to copy the packaged application(war file) to a server.

4) Simple to scale

   Simple to scale horizontally by running multiple copies behind a load balancer.

Drawbacks of Monolithic Architecture

1) Large and Complex Application

   It increase the size of the applications.

   It become complex to understand and modify such applications.

   As result development slows down and modularity breaks down over the time.

   Moreover it is difficult to understand how to currently implement the change due to that quality of code will decline over the time.

2) Slow Development

   As application and respective teams grows. The application really become difficult to understand and modify.

   Due to large base code which leads to slower the IDE which makes programmers less productive.

3) Blocks Continenous development

   In order to update one component/service. we need to re-deploy the entire application which interrupts the background.

   There is also a chance ,the components which never have been updated failed to start correctly. As result risk associated with redeployment increases which discourage the continueous development.

4) Unscalable

   We can't create instances for a perticular service.

   We need to create instance for entire services present in a monolithic application.

5) Unreliable

   Every service/component in monolethic application is tightly coupled.

   If any one of the service/component goes down the entire system failed to run.

   Moreever,A bug in any component/service potentially bring down entire process.

6)Inflexible

   It is very difficult to adopt new frameworks and languages.

   ex:

   Microservices can't communicate eachother. If they written in different languages.

MicroService Architecture
Microservices are the small services that work together
The microservice defines an approach to the architecture that divides an application into a pool of loosely coupled services that implements business requirements.
In Microservice architecture, Each service is self contained and implements a single bussiness capability.
The microservice architectural style is an approach to develop a single application as a suite of small services.It is next to Service-Oriented Architecture (SOA).
Each microservice runs its process and communicates with lightweight mechanisms.
These services are built around business capabilities and independently developed by fully automated deployment machinery.

Advantages of Microservice Architecture
1) Independent Development
Each microservice can be developed independently.
A single development team can build test and deploy the service.
2) Independent Deployment
we can update the service without redeploying the entire application.
Bug release is more manageable and less risky.
3) Fault Tolerance
If service goes down, It won't take entire application down with it.
4) Mixed Technology Stack
It is used to pick best technology which best suitable for our application.
5) Granular Scaling
In Granular scaling, services can scaled independently Instead of entire application.

customer micro service
To develop any micro service we need to follow spring boot flow layered architecture.

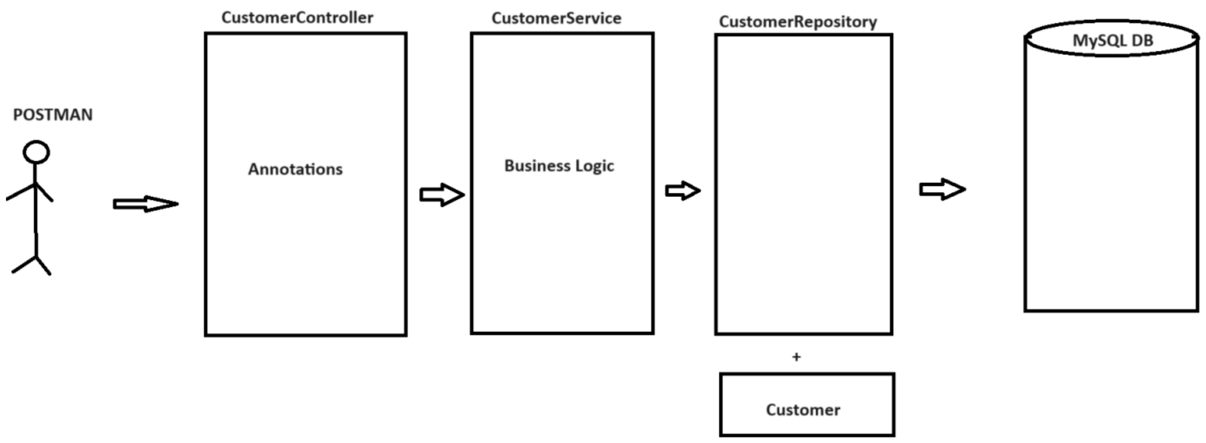

Diagram: sb5.1

step1:

Create a "customer-service" project.
starters:
- spring reactive web
- spring Data JPA
- Project Lombok
- mysql driver

step2:

Create a "demo" schema inside mysql database.
ex:
```
MYSQL> create schema demo;
MYSQL> use demo;
```

<u>Project structure</u>
```
customer-service
|
|-----src/main/java
|          |
           |-----com.ihub
           |          |
           |          |----CustomerMicroserviceApplication
           |
           |-----com.ihub.controller
           |          |
           |          |----CustomerController.java (controller class)
           |
           |-----com.ihub.entity
           |          |
           |          |----Customer.java (POJO class)
           |
           |-----com.ihub.service
           |          |
           |          |----CustomerService.java  (service class)
           |
           |-----com.ihub.repository
           |          |
           |          |----CustomerRepository.java (interface)
           |

|-------src/main/resources
|          |
|          |-----application.yml
           |
|
|-------pom.xml
|
```

step3:
Implement the logic as per the requirement.

Customer.java
```java
package com.ihub.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Customer {

	@Id
	@Column(length =6)
	private int custId;

	@Column(length =12)
	private String custName;

	@Column(length=12)
	private String custAddress;

}
```

<u>CustomerRepository.java</u>
```java
package com.ihub.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.ihub.entity.Customer;
```

```
public interface CustomerRepository extends JpaRepository<Customer, Integer> {

}
```

CustomerService.java

```
package com.ihub.www.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.data.repository.config.CustomRepositoryImplementationDetector;
import org.springframework.stereotype.Service;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;

import com.ihub.www.model.Customer;
import com.ihub.www.repo.CustomerRepository;

@Service
public class CustomerService
{
	@Autowired
	CustomerRepository customerRepository;


	public  Customer  addCustomer(Customer customer)
	{
		return customerRepository.save(customer);
	}

	public List<Customer> getAllCustomers()
	{
		return customerRepository.findAll();
	}

	public  Customer getCustomerById(int custId)
	{
		return  customerRepository.findById(custId).get();
	}
	public String updateCustomer(Customer customer)
	{
		Customer cust=customerRepository.findById(customer.getCustId()).get();

		cust.setCustName(customer.getCustName());
		cust.setCustAdd(customer.getCustAdd());

		customerRepository.save(cust);

		return "Record updated";
	}

	public String deleteCustomer(int custId)
	{
		Customer cust=customerRepository.findById(custId).get();
		customerRepository.delete(cust);

		return "Record Deleted";
	}
}
```

<u>CustomerController.java</u>

```
package com.ihub.www.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.service.annotation.PutExchange;

import com.ihub.www.model.Customer;
import com.ihub.www.service.CustomerService;

@RestController
@RequestMapping("/customer")
public class CustomerController
{
	@Autowired
	CustomerService customerService;

	@PostMapping("/add")
	public  Customer  addCustomer(@RequestBody  Customer customer)
	{
		return customerService.addCustomer(customer);
	}

	@GetMapping("/fetch")
	public List<Customer> getAllCustomers()
	{
		return customerService.getAllCustomers();
	}
	@GetMapping("/fetch/{custId}")
	public  Customer getCustomerById(@PathVariable int custId)
	{
		return customerService.getCustomerById(custId);
	}

	@PutMapping("/update")
	public String updateCustomer(@RequestBody Customer customer)
	{
		return customerService.updateCustomer(customer);
	}

	@DeleteMapping("/delete/{custId}")
	public String deleteCustomer(@PathVariable int custId)
	{
		return customerService.deleteCustomer(custId);
	}
}
```

<u>application.yml</u>

```
server:
  port: 9090



spring:
  application:
    name: CUSTOMER-SERVICE

  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/demo
    username: root
    password: root
  jpa:
    hibernate.ddl-auto: update
    generate-ddl: true
    show-sql: true
```

step3:

Run the spring boot application and get check below url.

<u>METHODS</u>	<u>URL</u>
GET	http://localhost:9001/customer/fetch
GET	http://localhost:9001/customer/fetch/101
POST	http://localhost:9001/customer/add

```
> body
        >raw

{
        "custId":101,
        "custName":"Alex",
        "custAdd":"Chicago"
}
```

PUT	http://localhost:9001/customer/update
DELETE	http://localhost:9001/customer/delete/101

Exception Handling in Spring Boot

If we give/pass wrong request to our application then we will get Exception.

ex: http://localhost:9090/fetch/102

Here '102' record is not available so immediately our controller will throw below exception.

ex:

```
{
   "timestamp": "2021-02-14T06:24:01.205+00:00",
   "status": 500,
   "error": "Internal Server Error",
   "path": "/fetch/102"
}
```

Handling exceptions and errors in APIs and sending the proper response to the client is good for enterprise applications.

In Spring Boot Exception handling can be performed by using Controller Advice.

@ControllerAdvice

The @ControllerAdvice is an annotation is used to to handle the exceptions globally.

@ExceptionHandler

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

<u>project structure</u>

```
customer-service
|
|----src/main/java
|         |
          |---com.ihub.www
                    |
                    |---CustomerServiceApplication.java

          |
          |---com.ihub.www.controller
                    |
                    |---CustomerController.java

          |---com.ihub.www.service
                    |
                    |---CustomerService.java

          |---com.ihub.www.repo
                    |
                    |----CustomerRepository.java(Interface)

          |---com.ihub.www.model
                    |
                    |----Customer.java


|         |---com.ihub.www.exception
|                   |
|                   |---ErrorDetails.java(POJO)
|                   |---ResourceNotFoundException.java
|                   |---GlobalExceptionHandler.java

|-----src/main/resources
|                   |
                    |---application.properties
|
|----pom.xml
```

}

step1: Use the existing project i.e customer-service.

step2: Create a com.ihub.www.exception package inside "src/main/java".

step3: Create ErrorDetails.java file inside "com.ihub.www.exception" pkg.

ErrorDetails.java

```
package com.ihub.www.exception;
import java.util.Date;
public class ErrorDetails
{
	private Date timestamp;
	private String message;
	private String details;


	public ErrorDetails(Date timestamp, String message, String details) {
		super();
		this.timestamp = timestamp;
		this.message = message;
		this.details = details;
	}

	public Date getTimestamp() {
		return timestamp;
	}
	public void setTimestamp(Date timestamp) {
		this.timestamp = timestamp;
	}
	public String getMessage() {
		return message;
	}
	public void setMessage(String message) {
		this.message = message;
	}
	public String getDetails() {
		return details;
	}
	public void setDetails(String details) {
		this.details = details;
	}
}
```

step4:	Create ResourceNotFoundException.java file inside "com.ihub.www.exception" pkg.

<u>ResourceNotFoundException.java</u>
```
package com.ihub.www.exception;

public class ResourceNotFoundException extends RuntimeException
{
	public ResourceNotFoundException(String msg)
	{
		super(msg);
	}
}
```

step5:	Create a GlobalExceptionHandler.java file inside
	"com.ihub.www.exception" pkg.

<u>GlobalExceptionHandler.java</u>
```
package com.ihub.www.exception;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;

@ControllerAdvice
public class GlobalExceptionHandler
{

	@ExceptionHandler(ResourceNotFoundException.class)
	public ResponseEntity<?> handleResourceNotFoundException
	(ResourceNotFoundException exception,WebRequest request )
	{
		ErrorDetails errorDetails=new ErrorDetails(new
Date(),exception.getMessage(),request.getDescription(false));
		return new ResponseEntity<>(errorDetails,HttpStatus.NOT_FOUND);
	}

	//handle global exception
		@ExceptionHandler(Exception.class)
		public ResponseEntity<?> handleException
```

```
                    (Exception exception,WebRequest request )
                    {
                              ErrorDetails              errorDetails=new              ErrorDetails(new
Date(),exception.getMessage(),request.getDescription(false));
                              return                                                          new
ResponseEntity<>(errorDetails,HttpStatus.INTERNAL_SERVER_ERROR);
                    }
}
```

step6: Now add ResourceNotFoundException to CustomerService.

<u>CustomerService.java</u>

```
package com.ihub.www.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.ihub.www.exception.ResourceNotFoundException;
import com.ihub.www.model.Customer;
import com.ihub.www.repo.CustomerRepository;

@Service
public class CustomerService
{
          @Autowired
          CustomerRepository customerRepository;

          public Customer addCustomer(Customer customer)
          {
                    return customerRepository.save(customer);
          }
          public List<Customer> getAllCustomer()
          {
                    return customerRepository.findAll();
          }

          public Customer getCustomer(int custId)
          {
                    return customerRepository.findById(custId)
                                        .orElseThrow(()-> new ResourceNotFoundException("ID NOT
FOUND"));
```

```
	}

	public String updateCustomer(Customer customer)
	{
		Customer cust=customerRepository.findById(customer.getCustId()).get();

		cust.setCustName(customer.getCustName());
		cust.setCustAdd(customer.getCustAdd());

		customerRepository.save(cust);

		return "Record updated";
	}

	public String deleteCustomer(int custId)
	{
		Customer customer=customerRepository.findById(custId)
		.orElseThrow(()->new ResourceNotFoundException("Id Not Found for
Delete"));

		customerRepository.delete(customer);

		return "Record is deleted";
	}
}
```

step7: Relaunch the spring boot application.

step8: Test the application by using below request url.
ex:
http://localhost:9090/fetch/102

step9: Here exception will display in below format.
ex:

```
	{
			"timestamp": "2023-03-27T23:04:03.181+00:00",
			"message": "ID NOT FOUND",
			"details": "uri=/fetch/102"eureka
	}
```

Types of API's

PIs are broadly accepted and used in web applications.

There are four different types of APIs commonly used in web services.

1) public API

A public API is open and available for use by any outside developer or business.

2) partner API

A partner API, only available to specifically selected and authorized outside developers or API consumers, is a means to facilitate business-to-business activities.

3) private API / Internal APIs

An internal or private API is intended only for use within the enterprise to connect systems and data within the business**.**

4) composite API

omposite APIs generally combine two or more APIs to craft a sequence of related or interdependent operations.

Eureka Server

This server holds information about the client service applications.
Each microservice registers into Eureka server and eureka server knows all client applications running on each port and IP address.
This server is also known as discovery server.

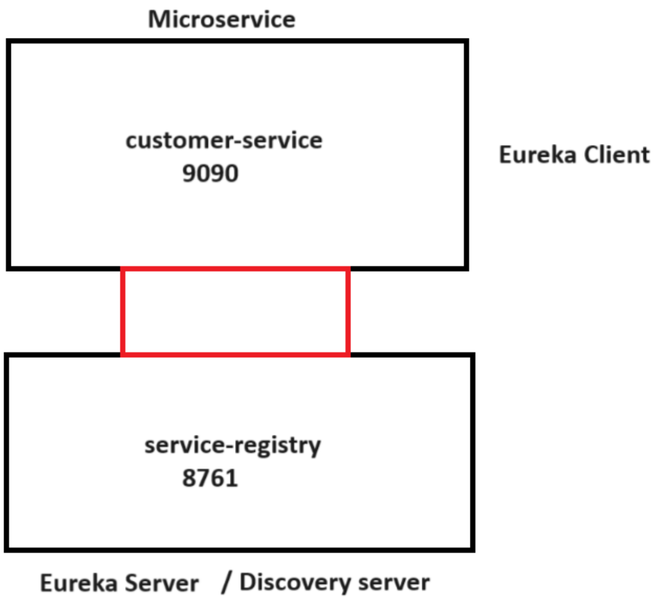

Diagram:

step1: Add Eureka Client dependency in "customer-service" project.
ex:
starter
Eureka Discovery client.

step2: Create a "service-registry" project to register all microservices.
Here "service-registry" is a Eureka Server and microservices are Eureka Clients.

```
        > service-registry
                starter
                        > Eureka Server.
```

step3: Add "@EnableEurekaServer" annotation in main spring boot application.

ServiceRegisterApplication.java

```
package com.ihub;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class ServiceRegisterApplication {

	public static void main(String[] args) {
		SpringApplication.run(ServiceRegisterApplication.class, args);
	}

}
```

step4: Add port number and set register for Eureka service as false.

application.yml

```
server:
  port: 8761

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

step5: Open the "customer-service" application.yml and add
register with eureka as true.

application.yml

```
server:
  port: 9001



spring:
  application:
    name: CUSTOMER-SERVICE

  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/demo
    username: root
    password: root
  jpa:
    hibernate.ddl-auto: update
    generate-ddl: true
    show-sql: true

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    hostname: localhost
```

step6: Now run all two projects.
First run service-registry then customer-service.
First run eureka server then eureka client.

step7: Check the output in below url's.
ex:
http://localhost:8761/

Spring Cloud API Gateway

Spring Cloud Gateway aims to provide a simple, effective way to route to API's and provide cross cutting concerns to them such as security,monitoring/metrics, authentication, autherization ,adaptor and etc.

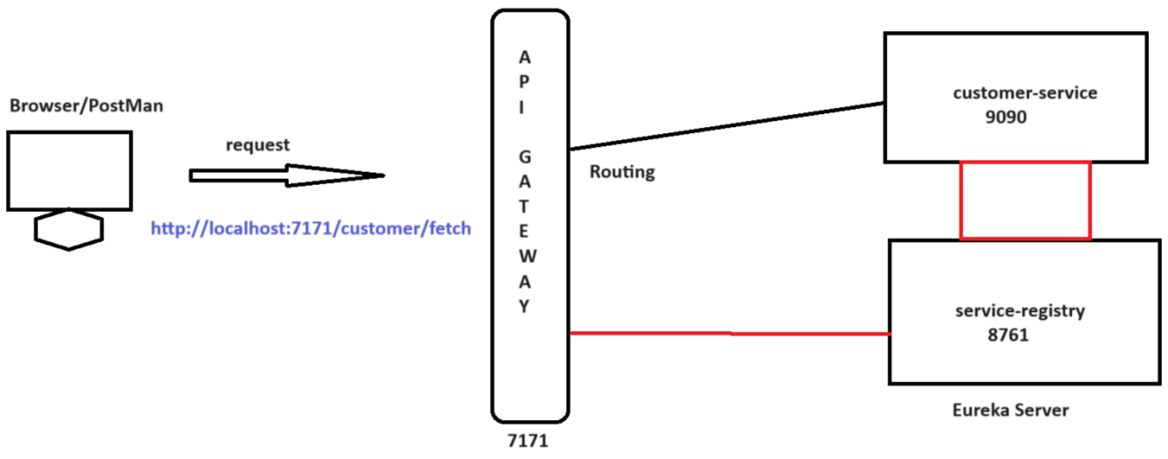

Diagram: sb6.2

step1: Create a "cloud-apigateway" project in STS.

        starters:
                eureka Discovery client
                Spring boot actuators
                spring reactive web

step2: Add spring cloud dependency in pom.xml file.
ex:

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
        <version>3.1.1</version>
</dependency>
```

step3: Add "@EnableEurekaClient" annotation on main spring boot application.

<u>CloudApigatewayApplication.java</u>

```
package com.ge;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class CloudApigatewayApplication {

        public static void main(String[] args) {
                SpringApplication.run(CloudApigatewayApplication.class, args);
        }

}
```

step4: Register port number, set application name,and configure
all microservices for routing in application.yml file.

<u>application.yml</u>

```
server:
  port: 7171

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    hostname: localhost

spring:
  application:
    name: API-GATEWAY
  cloud:
    gateway:
      routes:
        - id: CUSTOMER-SERVICE
          uri: lb://CUSTOMER-SERVICE
          predicates:
            - Path=/customer/**
```

step5: Now Run the following applications sequentially.
"service-registry"
"customer-service"
"cloud-apigateway".

step6: Test the applications by using below urls.
ex:
http://localhost:9191/customer/fetch/101
http://localhost:9191/customer/fetch

Spring Cloud Hystrix
Hystrix is a fault tolerance library provided by Netflix.
Using Hystrix we can prevent Deligation of failure from one service to another service.
Hystrix internally follows Circuit Breaker Design pattern.
In short circuit breaker is used to check availability of external services like web service call, database connection and etc.

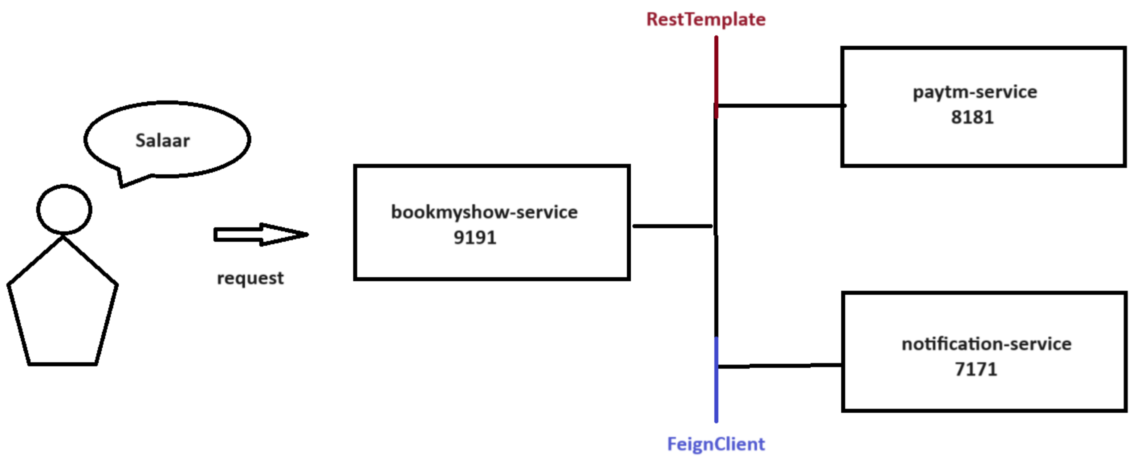

Diagram: sb8.1

notification-service

step1: create a "notification-service" project in STS.
Starter: Spring Web.

step2: Add the following code in main spring boot application.

NotificationServiceApplication.java
```
package com.ihub.www;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
@RequestMapping("/notification")
public class NotificationServiceApplication {

	@GetMapping("/send")
	public String sendEmail()
	{
		return "Email sending method is called from notification-service";
	}
	public static void main(String[] args) {
		SpringApplication.run(NotificationServiceApplication.class, args);
	}

}
```

step3: convert application.properties file to application.yml file.

step4: configure server port number in application.yml file.

application.yml
```
server:
  port: 7171
```

step5: Run "notification-service" project as spring boot application.

step6: Test the application with below request url.
ex:
http://localhost:7171/notification/send

paytm-service

step1: create a "paytm-service" project in STS.
Starter: Spring Web.

step2: Add the following code in main spring boot application.

PaytmServiceApplication.java

```java
package com.ihub.www;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
@RequestMapping("/paytm")
public class PaytmServiceApplication {

	@GetMapping("/pay")
	public String paymentProcess()
	{
		return "Payment Pocess method called in paytm-service";
	}

	public static void main(String[] args) {
		SpringApplication.run(PaytmServiceApplication.class, args);
	}

}
```

step3: convert application.properties file to application.yml file.

step4: configure server port number in application.yml file.

application.yml

```yaml
server:
  port: 8181
```

step5: Run "paytm-service" project as spring boot application.

step6: Test the application with below request url.
ex:
http://localhost:8181/paytm/pay

bookmyshow-service

step1: create a "bookmyshow-service" project in STS.
    Starter: Spring Web

step2: Add Spring Cloud Hystrix dependency in pom.xml file.
    ex:
```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
<version>2.2.10.RELEASE</version>
</dependency>
```
step3: Change <parent> tag inside pom.xml file for hystrix compatability.
    ex:
```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.3.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
</parent>
```
step4: Add the following code in main spring boot application.

BookmyshowServiceApplication

```
package com.ihub.www;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.hystrix.EnableHystrix;
import org.springframework.context.annotation.Bean;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@SpringBootApplication
@RestController
@EnableHystrix
public class BookmyshowServiceApplication {

        @Autowired
        RestTemplate restTemplate;
```

```
	@HystrixCommand(groupKey = "ihub" , commandKey = "ihub" ,fallbackMethod = "bookMyShowFallBack")
	@GetMapping("/book")
	public String bookShow()
	{
		String paytmServiceResponse=restTemplate.getForObject("http://localhost:8181/paytm/pay", String.class);
		String notificationServiceResponse=restTemplate.getForObject("http://localhost:7171/notification/send",String.class);

		return paytmServiceResponse+"\n"+notificationServiceResponse;
	}


	public static void main(String[] args) {
		SpringApplication.run(BookmyshowServiceApplication.class, args);

	}

	public String  bookMyShowFallBack()
	{
		return "service gateway failed";
	}

	@Bean
	public  RestTemplate    getRestTemplate() {

		return new RestTemplate();

	}

}
```
step5: convert application.properties file to application.yml file.
step6: configure server port number inside application.yml file.

<u>application.yml</u>
```
server:
  port: 9191
```

step7: Add spring core dependency inside pom.xml file.
ex:
```
		<dependency>
```

```
				<groupId>org.springframework</groupId>
				<artifactId>spring-core</artifactId>
				<version>5.3.17</version>
			</dependency>
```

step8:	Run the "bookmyshow-service" application as spring boot application.
step9:	Test the application by using below request url.
	ex:	http://localhost:9191/book
step10: Now stop any micro service i.e notification-service or paytm-service.
step11: Test the "bookmyshow-service" application by using below url.
	ex:
		http://localhost:9191/book
Note:	Here fallback method will execute with the help of Hystrix.

Spring Security
Spring Security is a framework which provides various security features like authentication, authorization to create secure Java Enterprise Applications.
It is a sub-project of Spring framework which was started in 2003 by Ben Alex.
Later on, in 2004, It was released under the Apache License as Spring Security 2.0.0.
This framework targets two major areas of application
1) Authentication
	It is a process of knowing and identifying the user that wants to access.
2) Authorization
	It is a process to allow authority to perform actions in the application.

Project structure

```
SpringSecurityApp
|
|----src/main/java
|        |
|        |----com.ge.www
|                  |
|                  |--SpringSecurityAppApplication.java
|        |
|        |----com.ge.www.controlller
|                  |
|                  |--HomeController.java
|
|---src/main/resources
|        |
|        |-----application.yml
|
|---src/test/java
|        |
|        |-----SBSpringSecurityApplicationTests.java
```

```
|---pom.xml
|
|
```

step1: create a spring starter project.
starters: spring web
spring security.

step2: create a Controller to accept the request.

<u>HomeController.java</u>

```java
package com.ge.www.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {

	@GetMapping("/msg")
	public String msg()
	{
		return "Welcome to Spring Security";
	}
}
```

step3: Configure server port number in application.properties file.

<u>application.yml</u>

```yaml
server:
  port: 9191
```

step4: Run the application as spring boot application.

step6: Test the application by using below url.
ex:
http://localhost:9191/msg

Note: When we hit the request ,we will get login page.
Default username is "user" and password we can copy from STS console.

step7: To change the default user and password we can use below properties in application.properties file.

application.yml

```
server:
  port: 9191

spring
  security:
    user:
      name=raja
      password=rani
```

step8: Relaunch the spring boot application.
step9: Test the application by using below url.
ex:
http://localhost:9191/msg

How can we convert spring project to jar file

step1: Make sure spring boot project is ready.
step2: Create a jar file for spring boot project.
ex: right click to project --> run as --> Maven build -->
Goals: package --> run.
step3: Check the jar file inside target folder of a spring boot project.

Q) What is the difference between WaterFall Model vs Agile Methodology?

Agile	**WaterFall Model**
It processes non-linear,incremental and interactive approach for a software design.	It processes linear and sequential approach for a software design.
At each incremental approach project is tested.	At the end whole project is tested.
A customer has early and frequent opportunities to look at the product and make decisions or changes to the project.	A customer only can see that product at the end of the project.
It is unstructured because it is not plan oriented.	It is structured because it is plan oriented.
Small projects can be implemented quickly.But large projects can't be estimated and implemented due to frequent changes.	All sort of projects can be estimated an implemented.