

Lecture 41: Layers and Tiers

How do you structure an application to support such operational requirements as maintainability, reusability, scalability and robustness? The answer lies in using Layers and Tiers? What different technologies Java provides to support layered or tiered architectures. The answer to these questions will remain our focus in this handout. A small case study will also be used to comprehend the concept of layers.

41.1 Layers vs. Tiers

Layers are merely logical grouping of the software components that make up the application or service, whereas Tiers refer to the physical residence of those layers.

In general,

Layers – represents the *logical view* of application

Tiers – represents *physical view* of application

However, both terms are used intractably very often. You must be confused what does logical & physical view mean? Let's elaborate layers and tiers further in detail to differentiate between them.

41.1.1 Layers

The partitioning of a system into layers such that each layer performs a specific type of functionality and communicates with the layer that adjoins it.

The separation of concerns minimizes the impact of adding services/features to an application. The application developed in layers also enables tiered distribution(discussed later). Furthermore easier maintenance, reuse of code, high cohesion & loose coupling sort of additional benefits are also enjoyed by the use of tiered architecture.

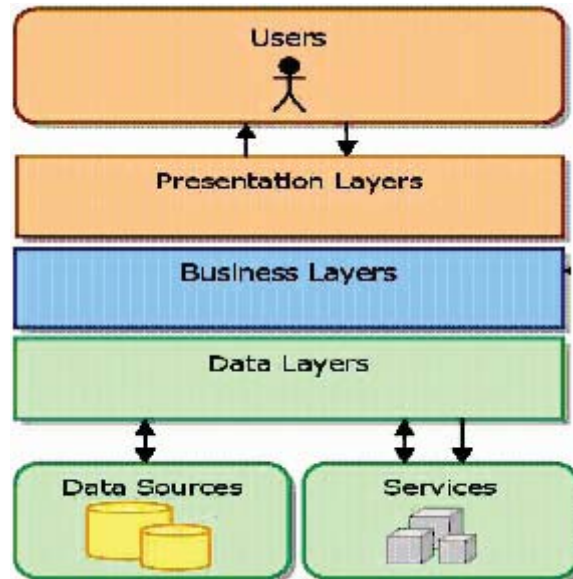
To begin with, layered architecture based on three layers. These are

- Presentation Layer
- Business Layer
- Data Layer

Note: However, there is no upper limit of number of layers an application can have. Each layer can also be further break down into several layers depending upon the requirements and size of the application.

Web Design and Development (CS506)

The figure given below shows a simplified view of an application and its layers.



As you can see in the figure, users can only interact with the presentation layer. The presentation layer passes the user request to the business layer, which further passes the request to the data layer. The data layer communicates with the data sources (like Database etc.) or other external services in order to accomplish the user request.

Let's discuss each layer's responsibility in detail:

41.1.1.1 Presentation Layer

It provides a user interface to the client/user to interact with the application. This is the only part of the application visible to client.

Its job list includes collecting user's input, validating user's input (on client side using JavaScript like technologies OR on server side), presenting the results of the request made by the user and controlling the screen flow (which page/view will be visible to the user).

41.1.1.2 Business Layer

Also called *application layer*, it is only concerned with the application specific functionality. It is used to implement business rules and to perform business tasks.

For example, in a banking system, this layer will provide the functionality of banking functions such as opening an account, transferring of balance from one account to another, calculation of taxes etc.

41.1.1.3 Data Layer

It is concerned with the management of the data & data sources of the system. Data sources can be database, XML, web services, flat file etc. Encapsulates data retrieval & storage logic For

Web Design and Development (CS506)

example, the address book application needs to retrieve all person records from a database to display them to the user.

41.1.2 Tiers

As mentioned, layers help in building a tiered architecture. Like layers, there is no restriction on using number of tiers. An application can be based on *Single-tier*, *Two-tier*, *Three-tier* or *N-Tier* (application which have more than three tiers). The choice of using a tiered architecture is contingent to the business requirements and the size of the application etc.

Tiers are physically separated from each other. Layers are spread across tiers to build up an application. Two or more layers can reside on one tier. The following figure presents a three-tier architectural view of an application.



The client tier represents the client machine where actually web browser is running and usually displays HTML. You can think of a Presentation as of two parts; one is on client side, for example, HTML. There is also a presentation layer that is used to generate the client presentation often called *server presentation*. We'll discuss about it later.

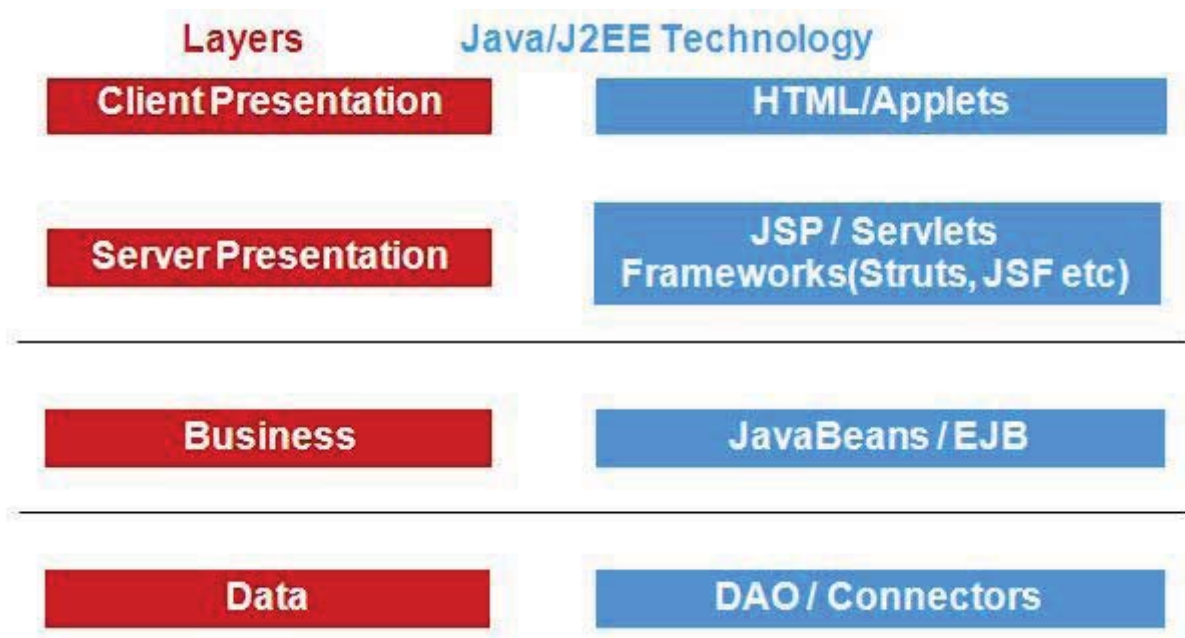
The server machine can consist on a single server machine or more. Therefore, it is possible *web server* is running on one server machine while *application server* on another. Web server is used to execute web pages like JSPs whereas application server is used to run special business objects like *Enterprise JavaBeans* (discussed later). The *web layer* and *applications server* can be on two separate machines or they can be on same tier as shown in the diagram.

The database server is often running on a separate tier, i.e. DB machine often called Enterprise information tier.

41.2 Layers Support in Java

The secret of wide spread use of Java lies in providing specific technology for each layer. This not only eases the development by freeing the programmer from caring about operational features but only reduces the production time of the software.

In the following figure, Presentation is bifurcated into two layers. These are *Client Presentation layer* and *Server Presentation Layer*. What the client sees in a browser forms the *client presentation layer* while the *server presentation layer* includes the Java technology components (JSP and Servlets etc.) that are used to generate the client presentation.



On the business layer, JavaBeans (also referred to as Plain Old Java Objects (POJO)) can be used. While moving towards a bigger architecture, the J2EE provides the special class that fits in the business layer i.e. Enterprise JavaBean (EJB).

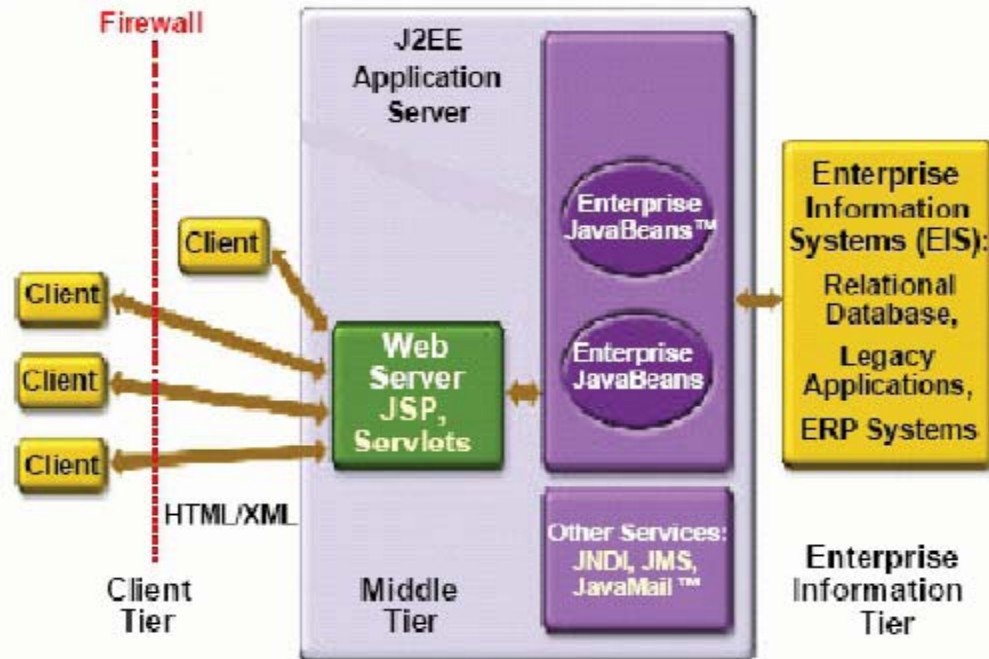
EJBs are special Java classes that are used to encapsulate business logic. They provide additional benefits in building up an application such as scalability, robustness, scalability etc.

On the data layer, Data Access Objects (DAO) can be used. Similarly, you can use *connectors*. There are other different specialized components provided in Java that ease the development of the data layer.

41.3 J2EE Multi-Tiered Applications

In a typical J2EE Multi-Tiered application, a client can either be a swing-based application or a web-based. As you can see in the following figure, clients can access the web server from behind the firewall as well.

Suppose, our client is HTML based. Client does some processing on HTML and transports it to web server. JSP and Servlets are possible technologies that can be used in a web server. However, there are some Frameworks such as JSF etc that can be used in a web server. The classes which form the presentation layer reside on web server and of course controllers are also used over here.



If web server, wants to perform some business process, it usually gets help from some business layer components. The business layer component can be a simple JavaBean (POJO) but in a typical J2EE architecture, EJBs are used. Enterprise JavaBeans interacts with the database or information system to store and retrieve data.

EJBs and JSP/Servlets works in two different servers. As you already know, JSP and Servlets runs in a *web server* where as EJBs requires an *application server*. But, generally application server contains the web server as well.

Application server including web server generally resides on a single tier (machine), which is often called middle tier. This tier stores and retrieves data from the *Enterprise Information Tier (EIS)* which is a separate tier. The response sends back to the client by the middle tier can be HTML, XML etc. This response can be seen on the separate tier know as client tier.

41.4 Case Study: Matrix Multiplication using Layers

Problem Statement

Calculate product of two matrices of order $2 * 2$

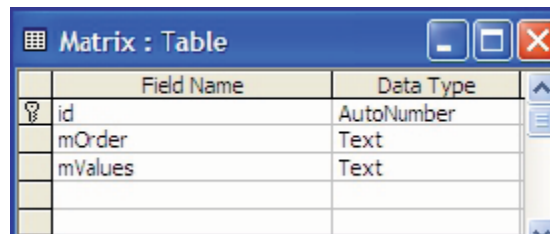
Result of multiplication should be stored in DB as well as shown to the user.

Format

- Input format

Web Design and Development (CS506)

- input will be in 4,2,6,5 format separated by commas where 4,2 represents entries of the first row
- **Display format**
 - Displays the matrix as a square
- **Storage format for DB**
 - Matrix will be stored as a string in the database along with the order of the matrix
 - The following figure shows the table design that will be used to store the results.

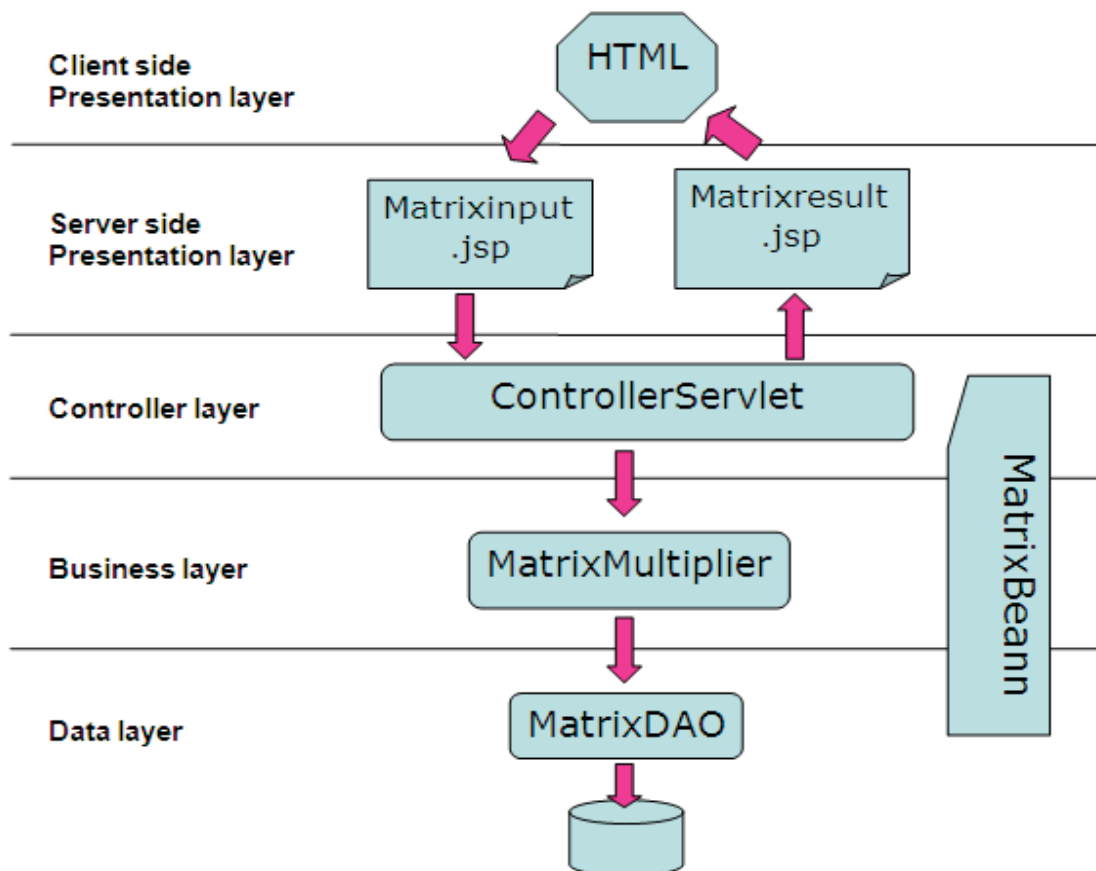


	Field Name	Data Type
PK	id	AutoNumber
	mOrder	Text
	mValues	Text

Layer by Layer View

A picture's worth than thousand words. Therefore, before jumping on to code, let's put a glance over layers that will be used in this small case study. The classes that will be used on each layer and what functionality each class will perform will also be discussed.

First, look on the following picture that will describe the whole story.



Web Design and Development (CS506)

The data layer has a class `MatrixDAO` that is used to save the matrix result into database. As mentioned in the problem statement, that resultant matrix should be saved in the database. So, `MatrixDAO` is used to accomplish that.

`MatrixDAO` called by the `MatrixMultiplier`, a business layer class. The functionality list of `MatrixMultiplier` includes:

- [-] Converting the user input string (e.g. 2,3,4,1) into a proper object i.e. a matrix data structure.
- Helps in calculating product of two matrices.

Controller layer's class `ControllerServlet` calls the `MatrixMultiplier`. This layer calls the various business methods (like multiplication of two matrices) of business layer class and got the resultant matrix. Furthermore, `ControllerServlet` sends the output to the `matrixresult.jsp` and receives the input from `matrixinput.jsp`.

The `MatrixBean` representing matrix data structure, as you can see in the figure is used across several layers. In fact, the object formed by `MatrixMultiplier` from a user input string is of `MatrixBean` type. It is used to transfer data from one layer to another.

First, look on the `MatrixBean` code given below:

MatrixBean

```
package bo;

import java.io.*;

public class MatrixBean implements Serializable{

    // a 2D array representing matrix
    public int matrix[ ][ ] ;

    // constructor
    public MatrixBean()
    {
        matrix = new int[2][2];
        matrix[0][0] = 0;
        matrix[0][1] = 0;
        matrix[1][0] = 0;
        matrix[1][1] = 0;
    }

    // setter that takes 4 int values and assigns these to array
    public void setMatrix(int w, int x, int y, int z)
    {
        matrix[0][0] = w;
        matrix[0][1] = x;
    }
}
```

```
matrix[1][0] = y;
matrix[1][1] = z;
}
// getter returning a 2D array
public int[ ][ ] getMatrix()
{
return matrix;
}
// used to convert 2D array into string
public String toString()
{

return matrix[0][0] + "," + matrix[0][1] + "," +
matrix[1][0] + "," +matrix[1][1] ;
}

} // end MatrixBean
```

matrixinput.jsp

This JSP is used to collect the input for two matrices in the form of string such as 2,3,5,8. The data will be submitted to ControllerServlet from this page.

```
<html>
<body>

<h2>
Enter Two Matrices of order 2 * 2 to compute Product
</h2>

<h3>
<%--
"controller" is an alias/URL pattern of ControllerServlet
--%>
<form name="matrixInput" action="controller" >
First Matrix:
<input type="text" name = "firstMatrix" /> E.g. 2,3,4,1
<br/>
Second Matrix:
<input type="text" name = "secondMatrix" />
<br/>
<input type = "submit" value = "Calculate Product" />
</form>
</h3>
</body>
</html>
```


ControllerServlet

This servlet acting as a controller receives the input from matrixinput.jsp. Furthermore, it will interact with the business layer class MatrixMultiplier to convert the string into a MatrixBean object, and to multiply two matrices.

```
package controller;

import bl.*;
import bo.* ;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;
public class ControllerServlet extends HttpServlet {

    // This method only calls processRequest()
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        processRequest(request, response);
    }

    // This method only calls processRequest()
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        processRequest(request, response);
    }

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // retrieving values from input fields of matrixinput.jsp
        String sMatrix1 = request.getParameter("firstMatrix");
        String sMatrix2 = request.getParameter("secondMatrix");

        // Creating MatrixMultiplier object
        MatrixMultiplier mm = new MatrixMultiplier();

        // Passing Strings to convertToObject() method of
        MatrixMultiplier
```

```
// convertToObject() is used to convert strings into MatrixBean
MatrixBean fMatrix = mm.convertToObject(sMatrix1);
MatrixBean sMatrix = mm.convertToObject(sMatrix2);

// passing MatrixBean's objects to multiply() method of
// MatrixMultiplier and receiving the product matrix in the form
// of MatrixBean
MatrixBean rMatrix = mm.multiply(fMatrix, sMatrix);

// saving results in database
mm.saveResult(rMatrix);

// storing the product of matrices into request, so that it can
// be
// retrieved on matrixresult.jsp
request.setAttribute("product", rMatrix);

// forwarding request to matrixresult.jsp
RequestDispatcher rd =
request.getRequestDispatcher("matrixresult.jsp");
rd.forward(request, response);
} // end processRequest()

} // end ControllerServlet
```

MatrixMultiplier

The business layer class that's primary job is to calculate product of two matrices given in the form of MatrixBean. This class also has a method convertToObject that takes a String and returns back a MatrixBean object. MatrixMultiplier will also interact with the data layer class MatrixDAO to store results in the database.

```
package bl;
import bo.*;
import dal.*;

public class MatrixMultiplier {

//constructor
public MatrixMultiplier( ) {

}
// used to convert a String (like 2,3,4,5) into a MatrixBean
object
public MatrixBean convertToObject(String sMatrix){

//splitting received string into tokens by passing "," as
//delimiter
```

```
String tokens[] = sMatrix.split(",");

//creating MatrixBean object
MatrixBean matrixBO = new MatrixBean();

// converting tokens into integers
int w = Integer.parseInt(tokens[0]);
int x = Integer.parseInt(tokens[1]);
int y = Integer.parseInt(tokens[2]);
int z = Integer.parseInt(tokens[3]);

// setting values into MatrixBean object by calling setter
matrixBO.setMatrix(w , x , y, z);

return matrixBO;

} // end convertToObject()

// used to multiply two matrices , receives two MatrixBean
objects
// and returns the product in the form of MatrixBean as well
public MatrixBean multiply(MatrixBean fMatrix , MatrixBean
sMatrix)
{
// creating MatrixBean object where product of the matrices will
// be
// stored
MatrixBean resultMatrix = new MatrixBean();

// retrieving two dimensional arrays from MatrixBeans object to
// perform multiplication
int matrixA[ ][ ] = fMatrix.getMatrix();
int matrixB[ ][ ] = sMatrix.getMatrix();
int matrixC[ ][ ] = resultMatrix.getMatrix();

// code to multiply two matrices
for (int i=0; i<2; i++) {
for (int j=0; j<2; j++) {
for (int k=0; k<2; k++) {
matrixC[i][j] += (matrixA[i][k] * matrixB[k][j]);
}
}
// storing the product from 2d array to MatrixBean object by
// calling setter
resultMatrix.setMatrix( matrixC[0][0], matrixC[0][1],
matrixC[1][0], matrixC[1][1] );
return resultMatrix;
}
```

```
} // end multiply()
// save results (MatrixBean containg product of two matrices)
//into
// database using DAO
public void saveResult( MatrixBean resultMatrix )
{
    MatrixDAO dao = null;
try{
dao = newMatrixDAO();}
catch(ClassNotFoundException e){}
catch(SQLException e){}
dao.saveMatrix(resultMatrix);
}
} // end MatrixMulitplier
```

MatrixDAO

As class name depicts, it is used to store product results into database. Let's look on the code to see how it is accomplished.

```
package dal;
import java.util.*;
import java.sql.*;
import bo.*;
public class MatrixDAO{
private Connection con;

// constructor
public MatrixDAO() throws ClassNotFoundException , SQLException
{
establishConnection();
}
// method used to establish connection with db
private void establishConnection() throws ClassNotFoundException
,SQLException
{
// establishing conection
class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String conUrl = "jdbc:odbc:MatrixDSN";
con = DriverManager.getConnection(conUrl);
}
// used to store MatrixBean into database after converting it to
// a String
public void saveMatrix(MatrixBean matrix){
try
{
```

```
String sql = "INSERT INTO Matrix(mOrder, mValues) VALUES (?,?)";

PreparedStatement pStmt = con.prepareStatement(sql);

// converting MatrixBean into String by calling toString()
String sMatrix = matrix.toString();

// setting order of matrix
pStmt.setString( 1 , "2*2" );

// setting matrix values in the form of string
pStmt.setString( 2 , sMatrix );

pStmt.executeUpdate();
}catch(SQLException sqlx){
System.out.println(sqlx);
}

} // end saveMatrix

// overriding finalize method to release acquired resources
public void finalize( ) {
try{
if(con != null){
con.close();
}
}catch(SQLException sex){
System.out.println(sex);
}
} // end finalize

} // end MatrixDAO class
```

matrixresult.jsp

Used to display resultant product of two matrices. The code is given below:

```
<!-- importing "bo" package that contains MatrixBean -->
<%@ page import="bo.*"%>

<html>

<body>

<h1>The resultant Matrix is </h1>
```

```
<%--
retrieving MatrixBean object from request, that was set on
ControllerServlet
--%>

<%
MatrixBean productMatrix =
(MatrixBean)request.getAttribute("product");

// retrieving values in 2d array so that it can be displayed
int matrix[][] = productMatrix.getMatrix() ;
%>

<%-- displaying MatrixBean's object values --%>

<TABLE>

<TR>
<TD> <%= matrix[0][0] %> </TD>
<TD> <%= matrix[0][1] %> </TD>
</TR>

<TR>
<TD> <%= matrix[1][0] %> </TD>
<TD> <%= matrix[1][1] %> </TD>
</TR>

</TABLE>

</body>

</html>
```

web.xml

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

<web-app>

<servlet>

<servlet-name> ControllerServlet </servlet-name>

<servlet-class> controller.ControllerServlet </servlet-class>

</servlet>
```

```
<servlet-mapping>
<servlet-name> ControllerServlet </servlet-name>
<url-pattern> /controller </url-pattern>
</servlet-mapping>
</web-app>
```

41.5 References:

- Java A Lab Course by Umair Javed.
- Java Passion by Sang Shin

Note: Coding exercises in working condition for this lecture are also available on “**Downloads**” section of LMS.

Lecture 42: Expression Language

Sun Microsystems introduced the Servlet API, in the later half of 1997, positioning it as a powerful alternative for CGI developers who were looking around for an elegant solution that was more efficient and portable than CGI (Common Gateway Interface) programming. However, it soon became clear that the Servlet API had its own drawbacks, with developers finding the solution difficult to implement, from the perspective of code maintainability and extensibility. It is in some ways, this drawback that prompted the community to explore a solution that would allow embedding Java Code in HTML - JavaServer Pages (JSP) emerged as a result of this exploration.

Java as the scripting language in JSP scares many people particularly web page designers which have enough knowledge to work with HTML and some scripting language, faced lot of difficulties in writing some simple lines of java code. Can we simplify this problem to ease the life of web designer? Yes, by using *Expression Language (EL)*.

JavaServer Pages Standard Tag Library (JSTL) 1.0 introduced the concept of the EL but it was constrained to only the JSTL tags. With JSP 2.0 you can use the EL with template text.

Note: - JSTL will be discussed in the following Handout.

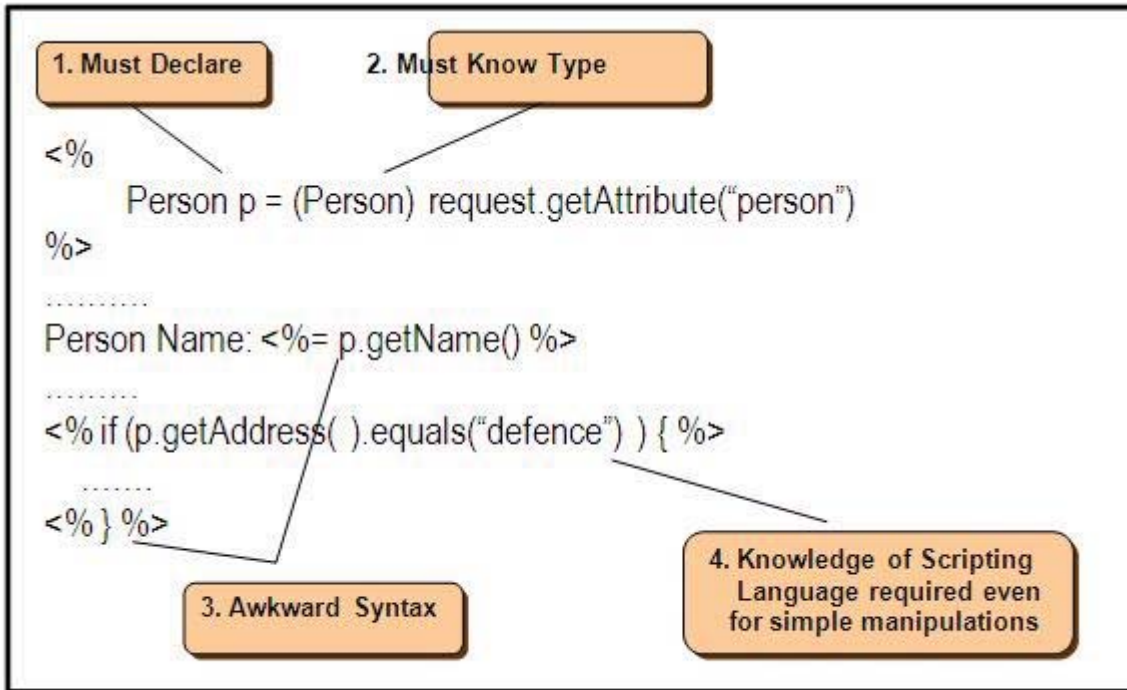
42.1 Overview

The Expression Language, not a programming or scripting language, provides a way to simplify expressions in JSP. It is a simple language that is geared towards looking up objects, their properties and performing simple operations on them. It is inspired form both the ECMAScript and the XPath expression language.

42.2 JSP Before and After EL

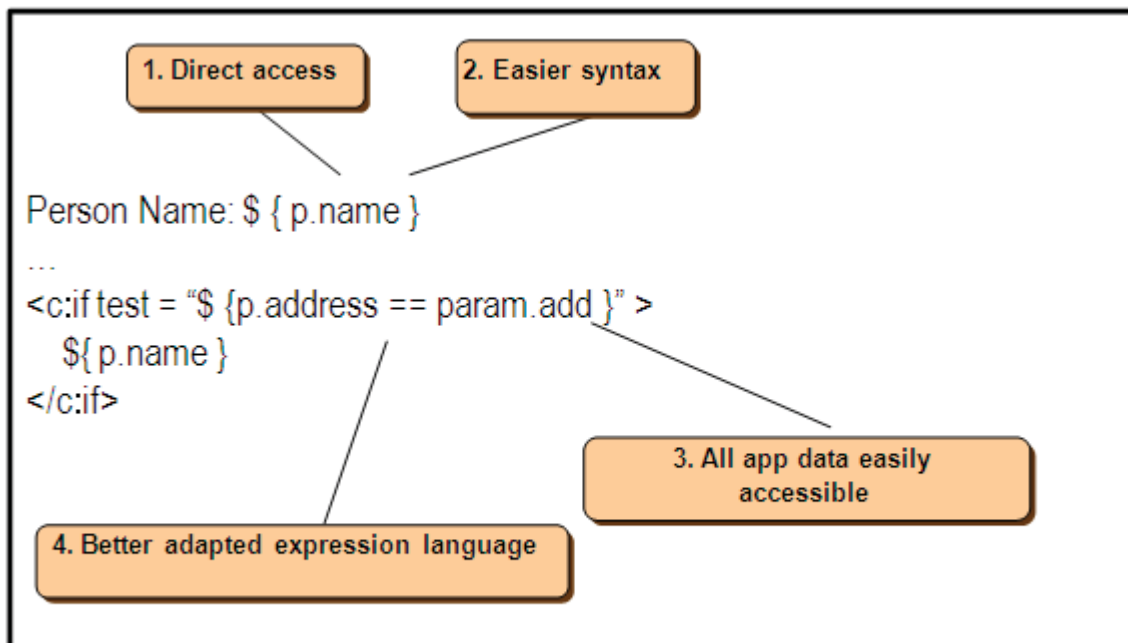
To add in motivational factor so that you start learning EL with renewed zeal and zest, a comparison is given below that illustrates how EL affects the JSPs.

The following figure depicts the situation of a JSP before EL. We have to declare a variable before using it, data type must be known in advance and most importantly have to use awkward syntax and many more. All these problems are highlighted in the following figure:



JSP before EL

Contrary to the above figure, have a look on the subsequent figure that gives you a hint how useful EL can be?



JSP After EL