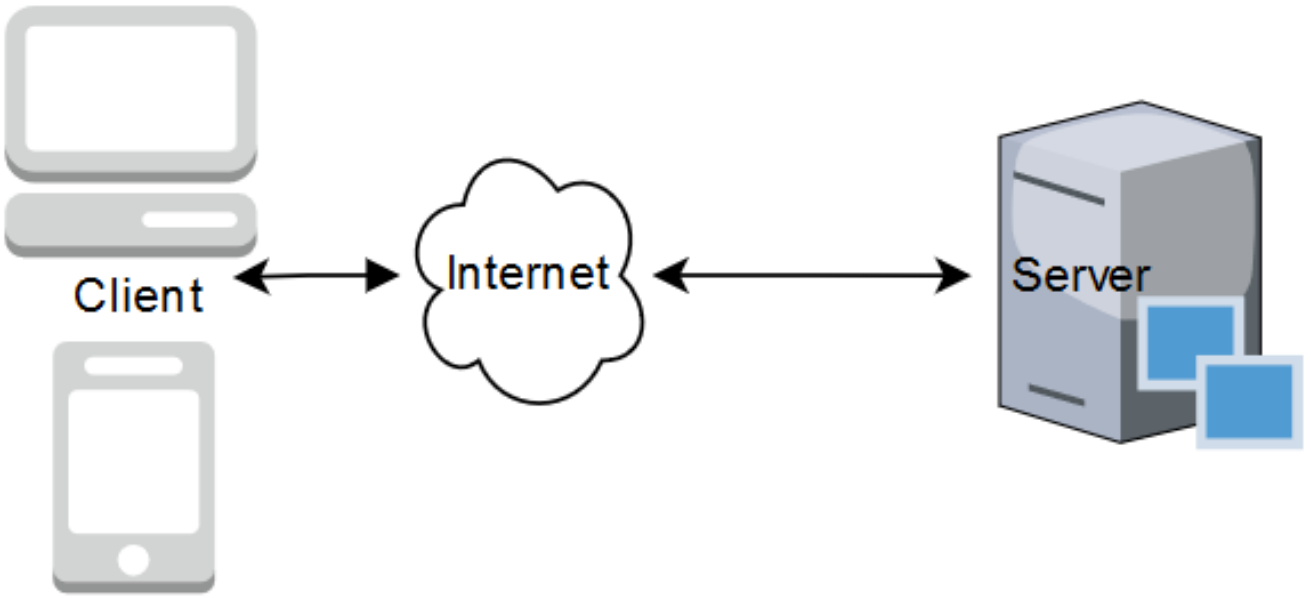
B.Sc. In Software Development. Year 4. Semester I. Enterprise Development. Introduction to JSP's.

LIMERICK INSTITUTE OF TECHNOLOGY
SCHOOL OF SCIENCE, ENGINEERING & I.T.
Department of Information Technology

Introduction to Web Development

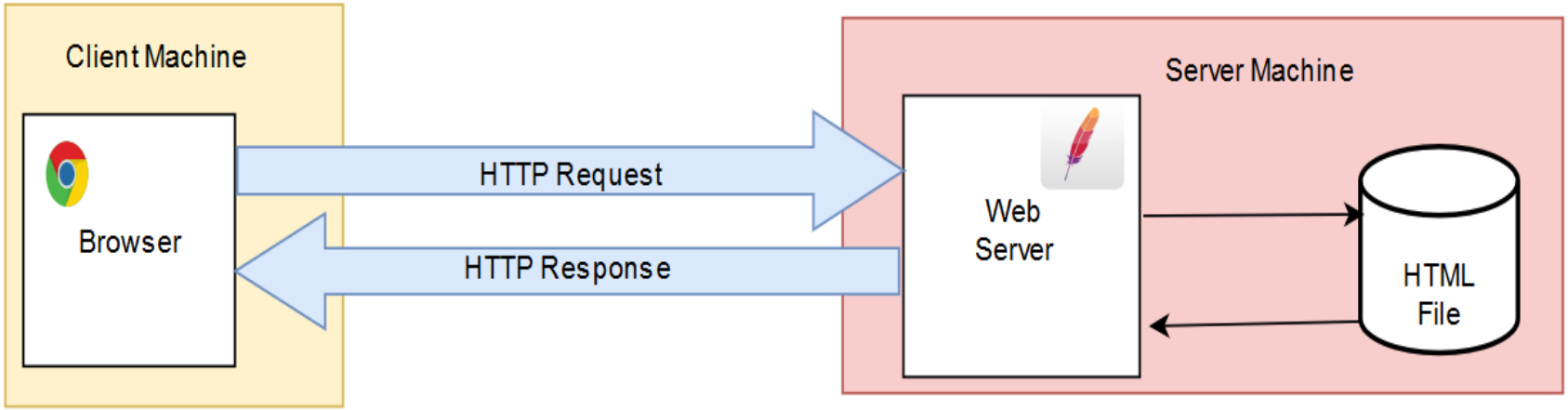
Basic components of a web application.



Introduction to Web Development

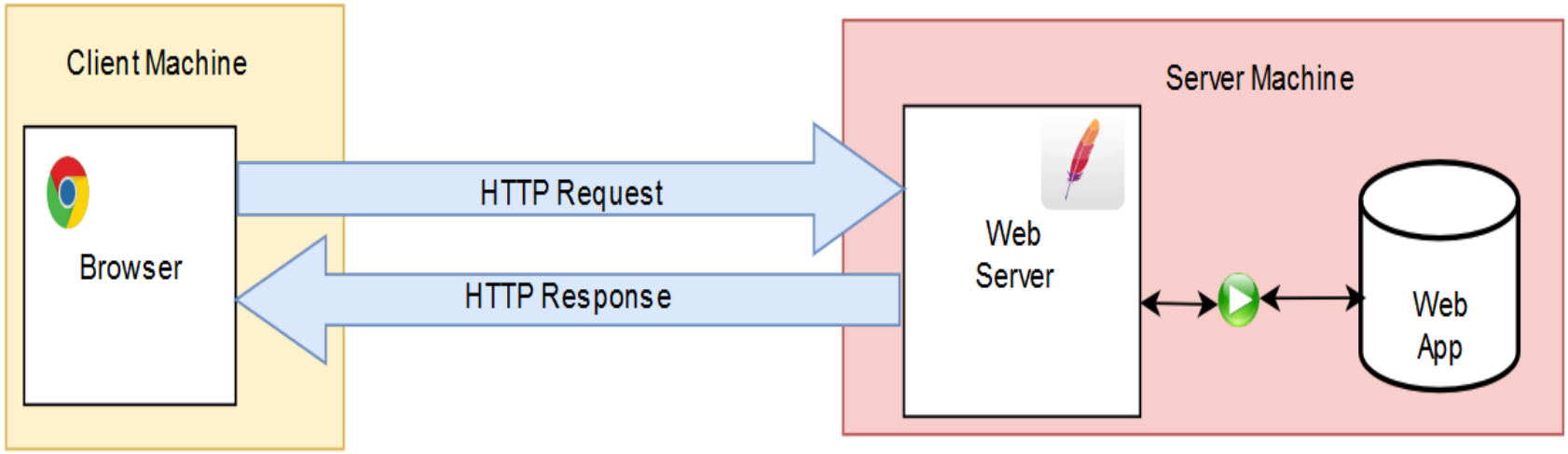
- This server computer runs web server software that enables it to send web pages to browsers.
- We will use Apache Tomcat (initially).
- Because most web applications work with data that's stores in a database, most servers run a DBMS.
- We will use MySQL.
- The DBMS does not have to run on the same machine as the web server.

Introduction to Web Development



Processing static web pages.

Introduction to Web Development



Processing dynamic pages.

Java Web Development

- In the early days of Java, the language received much attention for its ability to create Applets.
- Applets are not longer popular for a number of reasons.

Java Web Development

- To run a Java web application, the server machine must run two pieces of software. A web server (Apache/IIS etc) and a Servlet engine/container.
- The servlet engine/container (also called a JSP engine/container) allows the web server to run JSP's and Servlets.
- Oracle's Java Enterprise Edition (Java EE) specification describes how a JSP/Servlet should interact with a web server.
- Since all servlet/JSP engines must implement this specification all your servlets should work similarly.

Java Web Development

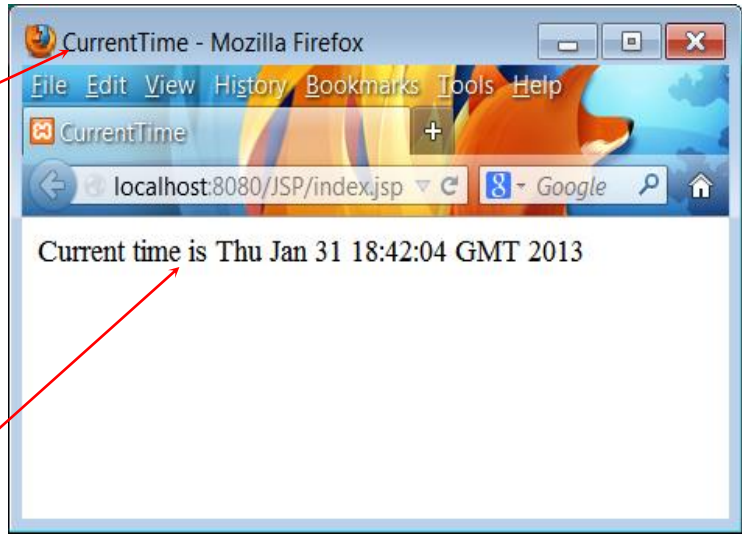
- May large websites also use a technology called Enterprise JavaBeans (EJBs).
- To use EJB's the server must also run an additional piece of software called an EJB server/container.
- There are benefits to using EJB's but they can greatly overcomplicate small and moderately sized sites.

Introduction to JSP's

- Java Server Pages (JSP) technology provides a simplified, fast way to create dynamic web content.
- JSP's are a mixture of HTML and Java.
- JSPs are compiled into Servlets by a JSP compiler.
- We'll look at Servlets in the future.

A Simple JSP

```
<!-- CurrentTime.jsp -->
<HTML>
<HEAD>
<TITLE>
CurrentTime
</TITLE>
</HEAD>
<BODY>
Current time is <%= new java.util.Date() %>
</BODY>
</HTML>
```



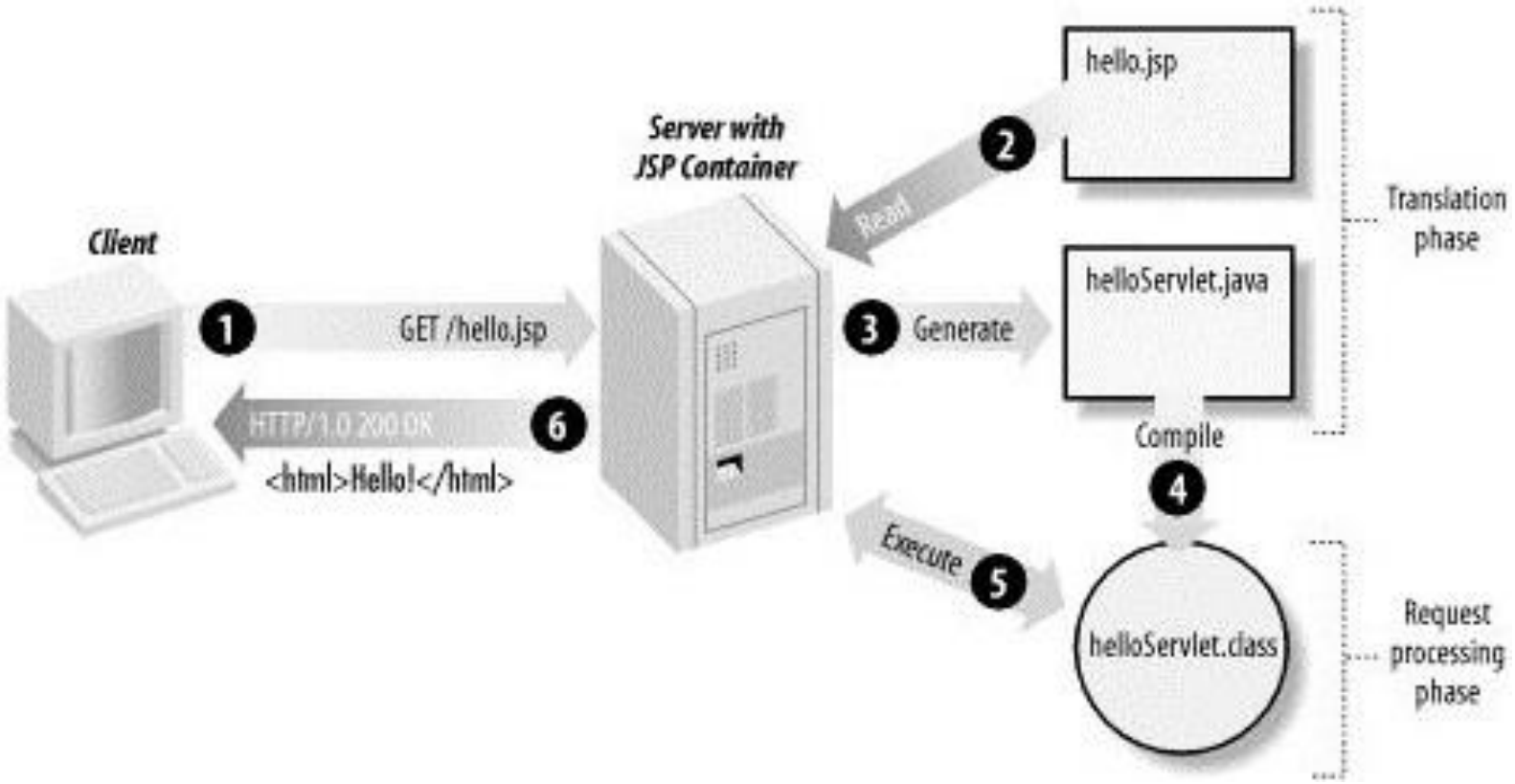
How a JSP Is Processed

- An HTTP request is sent to the web server.
- The web server will determine that it's a JSP request so it will call the JSP engine. This process is done for the pages that have a .jsp extension.
- The requested JSP page will be loaded from the disk and the JSP engine will convert it into a servlet.
- All the expressions are converted into proper Java code by the JSP engine.

How a JSP Is Processed

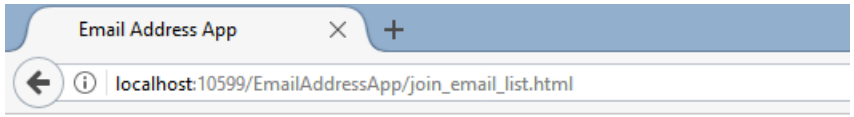
- The generated servlet is compiled and converted into a class. This class is then forwarded to servlet engine.
- Finally, this class is loaded and executed by the servlet engine. In the servlet execution process, HTML output is generated which is passed to web server.
- The web server is responsible for sending this HTML output back to the web client.
- As a last step, the web browser handles the HTML output and renders it on the web page

How a JSP Is Processed



Example 1: Working With Forms

HTML Page



Join Our Email List

To join our email list, enter your name and email address below,

First name: Alan

Last name: Ryan

Email address: alan.ryan@lit.ie

Submit

JSP Page

Thanks for joining our email list

Here is the information you entered:

First name:	Alan
Last name:	Ryan
Email address:	alan.ryan@lit.ie

To enter another email address, click on the Back button in your browser or the Return button shown below.

Return

Example 1: Working With Forms

```
8         <title>Email Address App</title>
9         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10    </head>
11    <body>
12        <h1>Join Our Email List</h1>
13
14        <p>To join our email list, enter your name and email address below, </p>
15
16        <form action="display_email_entry.jsp" method="get">
17            <table cellspacing="5" border="0">
18                <tr>
19                    <td align="right"> First name: </td>
20                    <td><input type="text" name="firstName"> </td>
21                </tr>
22
23                <tr>
24                    <td align="right"> Last name: </td>
25                    <td><input type="text" name="lastName"> </td>
26                </tr>
27
28                <tr>
29                    <td align="right"> Email address: </td>
30                    <td><input type="text" name="emailAddress"> </td>
31                </tr>
32                <tr>
33                    <td></td>
34                    <td><br><input type="submit" value="Submit"> </td>
35                </tr>
36
37
38
39            </table>
40        </form>
```

Example 1: Working With Forms

```
10
11          <%
12
13          String firstName = request.getParameter("firstName");
14          String lastName = request.getParameter("lastName");
15          String emailAddress = request.getParameter("emailAddress");
16
17
18
19          %>
20
21          <h1> Thanks for joining our email list </h1>
22
23          <p> Here is the information you entered: </p>
24
25          <table cellspacing="5" cellpadding="5" border="1">
26              <tr>
27                  <td align="right"> First name: </td>
28                  <td><%= firstName%> </td>
29              </tr>
30              <tr>
31                  <td align="right"> Last name: </td>
32                  <td><%= lastName%> </td>
33              </tr>
34
35              <tr>
36                  <td align="right"> Email address: </td>
37                  <td><%= emailAddress %> </td>
38              </tr>
39          </table>
40
41          <p>To enter another email address, click on the Back <br>
42             button in your browser or the Return button shown <br>
43             below. </p>
44
45          <form action="join_email_list.html" method="post">
46              <input type="submit" value="Return">
47          </form>
```

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet.

1. Expressions.
2. Scriptlets.
3. Declarations.

JSP Constructs: Expressions

- Used to insert a Java expression directly into the output. It has the following form:

No semicolon

```
<%= Integer.parseInt(num1) %>
```

- The expression is evaluated, converted into a `String`, and included into the output.

JSP Constructs: Scriptlets

- Allows you to write blocks of Java code inside the JSP.
- You place the code between <% and %> characters.

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

- Usually don't contribute any HTML
- Contain code that is executed every time the JSP is invoked.

JSP Constructs: Declarations

- Used for declaring methods or fields in the JSP.
- It has the following form:

  <%! Java method or field declaration %>

JSP Comments

- HTML comments have the following form:

  <!-- HTML Comment -->

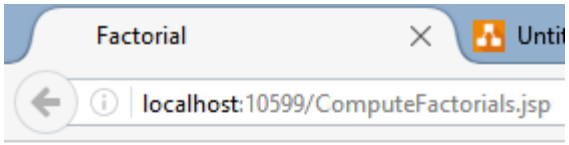
- If you don't want the comment appear in the resultant HTML file, use the following comment in JSP:

  <%-- JSP Comment --%>

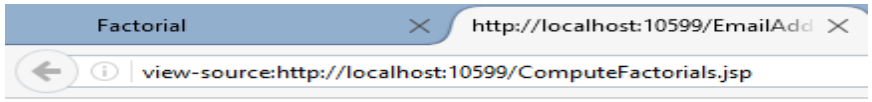
Putting it all Together

```
<HTML>
    <HEAD> <TITLE> Factorial </TITLE> </HEAD>

    <BODY>

        <!-- HTML Comment -->
        <%-- JSP Comment --%>


        <%  for (int i = 0; i <= 10; i++) {%>
        Factorial of <%= i%> is
        <%= computeFactorial(i)%> <br>
        <%  }%>

        <%! private long computeFactorial(int n) {
                if (n == 0) {
                    return 1;
                } else {
                    return n * computeFactorial(n - 1);
                }
            }
        %>

    </BODY>
</HTML>
```

Putting it all Together



Factorial of 0 is 1
Factorial of 1 is 1
Factorial of 2 is 2
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120
Factorial of 6 is 720
Factorial of 7 is 5040
Factorial of 8 is 40320
Factorial of 9 is 362880
Factorial of 10 is 3628800




```
<HTML>
    <HEAD> <TITLE> Factorial </TITLE> </HEAD>

    <BODY>

        <!-- HTML Comment -->




        Factorial of 0 is
        1 <br>

        Factorial of 1 is
        1 <br>

        Factorial of 2 is
        2 <br>

        Factorial of 3 is
        6 <br>

        Factorial of 4 is
        24 <br>

        Factorial of 5 is
        120 <br>

        Factorial of 6 is
        720 <br>

        Factorial of 7 is
        5040 <br>

        Factorial of 8 is
        40320 <br>

        Factorial of 9 is
        362880 <br>

        Factorial of 10 is
        3628800 <br>



    </BODY>
</HTML>
```

JSP Predefined Variables

For convenience, JSP provides a number of predefined variables from the Servlet environment that can be used with JSP expressions and scriptlets. Commonly used ones include:

1. request
2. response
3. out
4. session
5. application
6. config
7. pagecontext

JSP Predefined Variables: request

Represents the client's request, which is an instance of `HttpServletRequest`.

You can use it to access request parameters, HTTP headers such as cookies, hostname, etc.

```
<%
  String usersName =
     request.getParameter( "NameField" );
%>
```

JSP Predefined Variables: response

Represents the Servlet's response, which is an instance of HttpServletResponse.

You can use it to set response types and send output to the client. It can also be used to add new cookies or date stamps, HTTP status codes etc.

```
<%

response.setContentType("text/html");

%>
```

JSP Predefined Variables: out

The out implicit object is an instance of a javax.servlet.jsp.JspWriter object and is used to send content in a response.

You can use it to send character content to the client (browser).

```
<%
   out.println(“Welcome To My Website");
%>
```

JSP Predefined Variables: session

Represents the `HttpSession` object associated with the request, obtained from `request.getSession()`.

The session object is used to track client session between client requests

```
<%
    session = request.getSession( true );
%>
```

JSP Predefined Variables: application

The difference between session and application is that session is tied to one client, but application is for all clients to share persistent data.

```
<%
   application.setAttribute("objectName", "shared info");
   String s = (String) application.getAttribute("objectName");
   out.println(s);
%>
```

JSP Predefined Variables: config

This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.

```
<%

 String sName = config.getServletName();
 out.println("<br> the name is " + sName);

%>
```

JSP Predefined Variables: pageContext

Represents the `PageContext` object.

`PageContext` is a new class introduced in JSP to give a central point of access to other implicit JSP objects

```
<%
    session = pageContext.getSession();
%>
```

Example 2: Using Regular Classes With JSP's

- Any regular classes can be used by a JSP.
- In the next example we are going to use a `User` and `UserIO` class in conjunction with the first example.
- The `User` class is going to model a user of the application. It contains three instance variables (`firstName`, `lastName` and `emailAddress`).

Example 2: Using Regular Classes With JSP's

```
1   package business;
2
3
4   public class User {
5
6       private String firstName;
7       private String lastName;
8       private String emailAddress;
9
10      public User() {
11      }
12
13      public User(String firstName, String lastName, String emailAddress) {
14          this.firstName = firstName;
15          this.lastName = lastName;
16          this.emailAddress = emailAddress;
17      }
18
19      /**
20       * @return the firstName
21       */
22      public String getFirstName() {
```

etc..

Example 2: Using Regular Classes With JSP's

- The `UserIO` class contains one static method called add that writes the values contained in a `User` object to a text file.

- The method accepts two parameters:
    1. A `User` object.
    2. A `String` that provides the path for the file.

Example 2: Using Regular Classes With JSP's

```
1   package data;
2
3   import business.User;
4   import java.io.File;
5   import java.io.FileWriter;
6   import java.io.IOException;
7   import java.io.PrintWriter;
8
9
10  public class UserIO {
11
12      public static void add(User user, String filepath) throws IOException {
13
14          File file = new File(filepath);
15          PrintWriter out = new PrintWriter(new FileWriter(file, true));
16          out.println(user.getEmailAddress() + "|" + user.getFirstName() + "|" + user.getLastName());
17          out.close();
18
19      }
20
21  }
```

Example 2: Using Regular Classes With JSP's

The JSP from the first example now needs to be enhanced so that when the information is sent to the server (the JSP) it uses it to create a `User` object and then uses the add method in the `UserIO` class to write the user's details to a file.

The code for the reworked JSP is on the next slide.

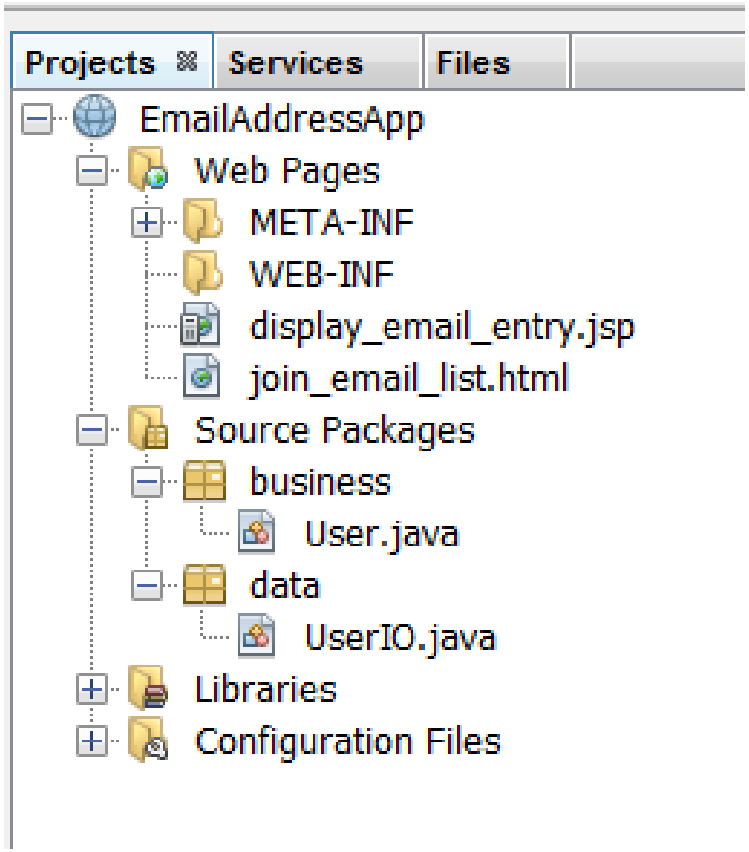
Example 2: Using Regular Classes With JSP's

```
4  <html>
5      <head>
6          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7          <title>Email Address App</title>
8      </head>
9      <body>
10         <%@ page import="business.User, data.UserIO" %>
11
12         <%
13
14         String firstName = request.getParameter("firstName");
15         String lastName = request.getParameter("lastName");
16         String emailAddress = request.getParameter("emailAddress");
17
18         ServletContext sc = this.getServletContext();
19
20         String path = sc.getRealPath("/WEB-INF/EmailList.txt");
21
22         User user = new User(firstName, lastName, emailAddress);
23
24         UserIO.add(user, path);
25
26         %>
27
28         <h1> Thanks for joining our email list </h1>
29
30         <p> Here is the information you entered: </p>
```

The rest of the JSP is omitted as it continues as before.

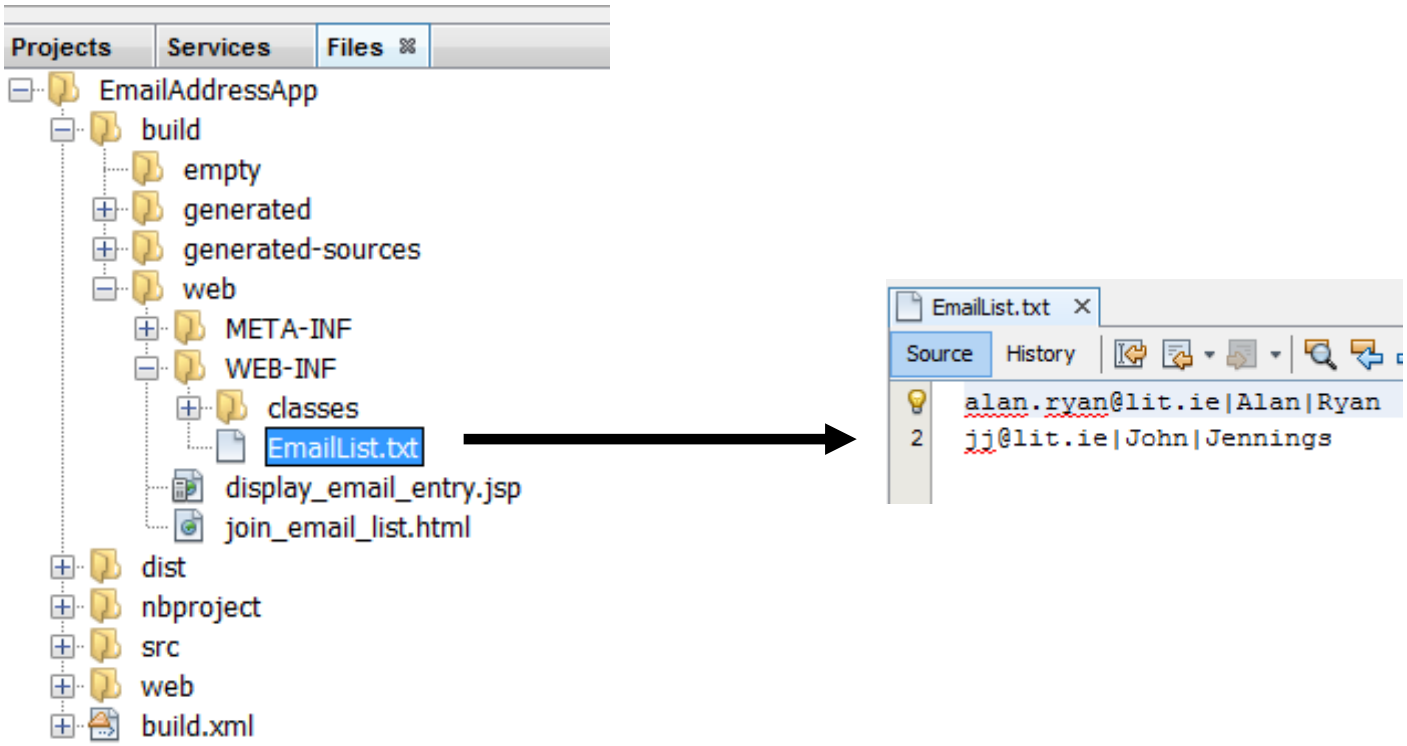
Analysing a Netbeans Web Project

The anatomy of my Netbeans project is as follows:



Analysing a Netbeans Web Project

As you can see the file is added to the project after the first `User` object is added.

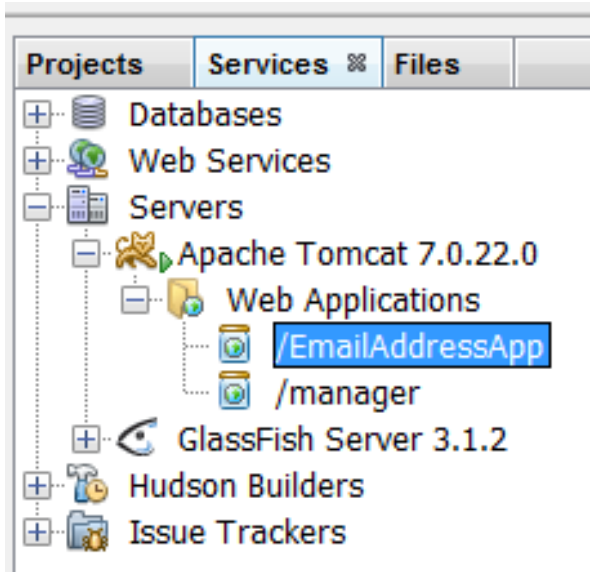


Analysing a Netbeans Web Project

Folder	Description
build\web	Contains the folders and files for the app after it has been built
dist	Contains the WAR file for the app
nbproject	Contains config files and build scripts
src	Contains the source code for the app.
test	Contains the source code for any automated tests
web	Contains the HTML, JSP, XML etc. files for the app.

Analysing a Netbeans Web Project

To view what (WAR) files are deployed on the server at any time, switch to the Services tab and expand the Servers node.



Note: When you run a web application, Netbeans automatically deploys the application on the server for you.

A word on working with files and paths

- When you work with files you typically use relative paths to refer to files that are available within the web application.
- Sometimes you need to get the real path for one of these files.
- To do that you use the technique that was used in the previous example.
- First you call the `getServletContext` method to get a `ServletContext` object.

A word on working with files and paths

- A `ServletContext` object is a configuration Object which is created when web application is started.
- It contains different initialization parameter that can be configured in web.xml.
- Once you have retrieved the `ServletContext` object you then call the `getRealPath` on it to get the real path for the specified file.

A word on working with files and paths

- When you use the `getRealPath` method, a front slash navigates to the root directory for the current application so:

```
getRealPath("/EmailList.txt");
```

- Specifies the EmailList.txt file in the web applications root directory.

- If you store a file in a directory that's web accessible, such as the root, then that file can be accessed by anyone who enters the correct URL.

References

Murach, J., (2014) *Murachs Java Servlets JSP*, 3rd edn. Mike Murach and Associates, Inc.

Y. Daniel Liang (2014) *Introduction to Java Programming, Comprehensive Version*, 10th edn. Pearson.

http://docs.oracle.com/javaee/6/tutorial/doc/