

# BAB 4

## JSP Dasar

### 4.1 Pengenalan

Pada bab sebelumnya, kita telah belajar bagaimana menghasilkan dynamic content untuk user kita dengan menggunakan teknologi Java melalui penggunaan servlets. Bagaimanapun juga, ketika pengembang Java dapat membuat sites dengan semacam dynamic content hanya dengan menggunakan servlet, ada beberapa kerugian dengan menggunakannya.

Pertama, hanya dengan menggunakan servlet untuk menghasilkan isi HTML tidak membutuhkan pemeliharaan yang banyak atau clean code. Strings yang digunakan untuk output HTML dapat menjadi sangat besar – spanning multiple lines – dan dapat dengan mudah pemeliharaannya menjadi rumit.

Kedua, hanya dengan menggunakan servlets untuk menghasilkan isi HTML dapat diasumsikan bahwa para pengembang telah mengenal Java & HTML dengan baik. Secara khusus dalam kasus sebuah organisasi yang besar, desiner site merupakan bagian dari kelompok programmer. Dengan melatih para desainer tentang pemahaman Java, atau untuk memiliki pengembang Java yang ahli dalam desain site HTML dapat dikonsumsi dan/atau sebuah aktifitas yang baik untuk sebuah organisasi.

Hal ini dimana teknologi JSP mulai digunakan.

### 4.2 Tujuan

#### 4.2.1 Apakah JSP itu?

Java Server Pages (JSP) merupakan sebuah teknologi servlet-based yang digunakan pada web tier untuk menghadirkan dynamic dan static content. JSP merupakan text-based dan kebanyakan berisi template text HTML yang digabungkan dengan spesifik tags dynamic content.

#### 4.2.2 Kenapa menggunakan JSP?

- Sejak JSPs merupakan dokumen text seperti HTML, para pengembang menghindari format dan manipulasi yang memungkinkan String yang sangat panjang untuk menghasilkan output. Content HTML sekarang tidak ditempelkan dengan berbagai macam kode dari Java. Hal ini membuatnya lebih mudah untuk dipelihara.
  - JSPs lebih dikenal oleh semua orang dengan pengetahuan dari HTML, hanya dengan mempelajari markup dynamic. Hal ini membuatnya mungkin untuk para desainer site untuk membuat template HTML dari sebuah site, dengan para pengembang memprosesnya suatu saat nanti untuk memasukkan tags yang menghasilkan dynamic content. Hal ini juga memudahkan dalam pengembangan web page.
  - JSPs memiliki built-in yang mendukung untuk penggunaan komponen software yang dapat digunakan kembali(JavaBeans). Hal ini tidak hanya membiarkan para pengembang menghindari kemungkinan menemukan kembali inti/kemudi dari tiap aplikasi, mempunyai software pendukung untuk memisahkan komponen software untuk handle logic promotes separation dari presentasi dan business logic.
  - JSPs, merupakan bagian solusi dari Java untuk pengembang aplikasi web, merupakan multi-platform yang tak terpisahkan dan dapat dijalankan pada berbagai container servlet
-

- yang compatible, dengan mengabaikan vendor atau sistem operasinya.
- Dalam kaitannya dengan cara kerja JSPs, mereka tidak membutuhkan kompilasi dari para pengembang. Kompilasi ini telah ada untuk kita pada kontainer servlet. Modifikasi JSPs dideteksi secara otomatis. Hal ini secara relatif membuatnya mudah untuk dibangun.

#### 4.2.3 Contoh JSP

```
<HTML>

<TITLE>Selamat Datang</TITLE>

<BODY>
  <H1> Greetings!</H1> <br>

  Terimakasih telah mengakses site kami. <br>

  Waktu hari ini <%= new java.util.Date()%>

</BODY>

</HTML>
```

`selamatdatang.jsp`

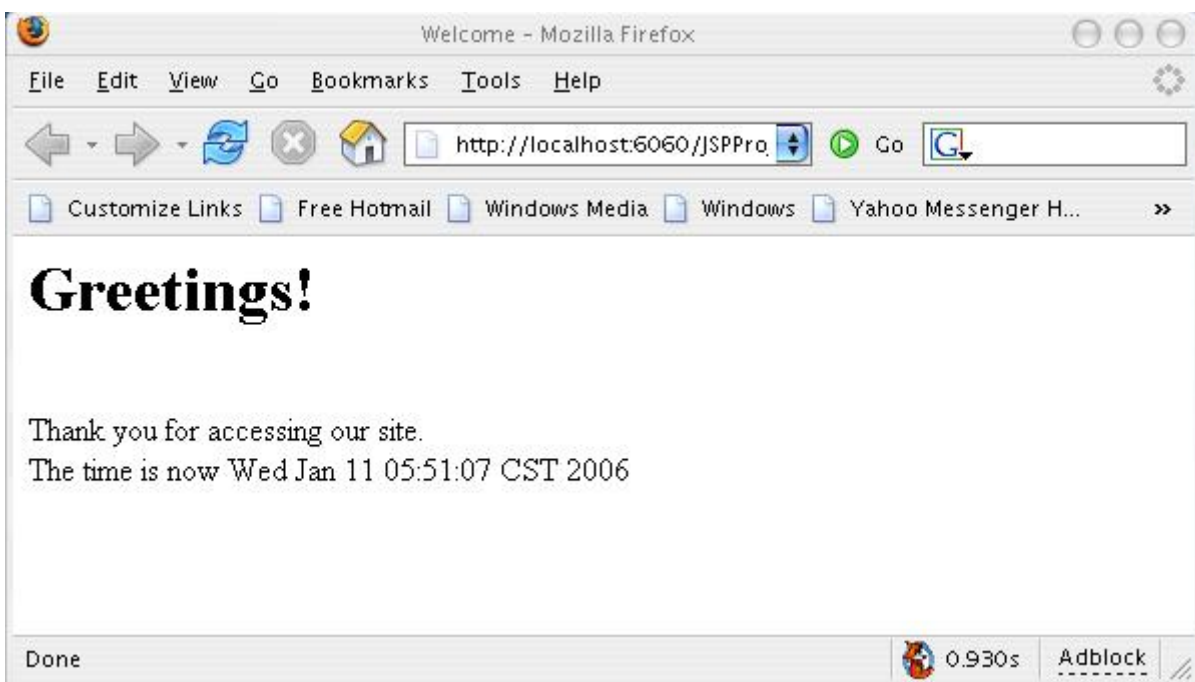
Gambar diatas adalah file JSP yang simple yang melakukan penyambutan untuk site user sebagai penginformasian tanggal dan waktu saat ini untuk user.

Dari contoh diatas, kita dapat melihat bahwa file JSP merupakan sebagian besar dari HTML. Hanya pada bagian ini yang beda :

```
<%= new java.util.Date()%>
```

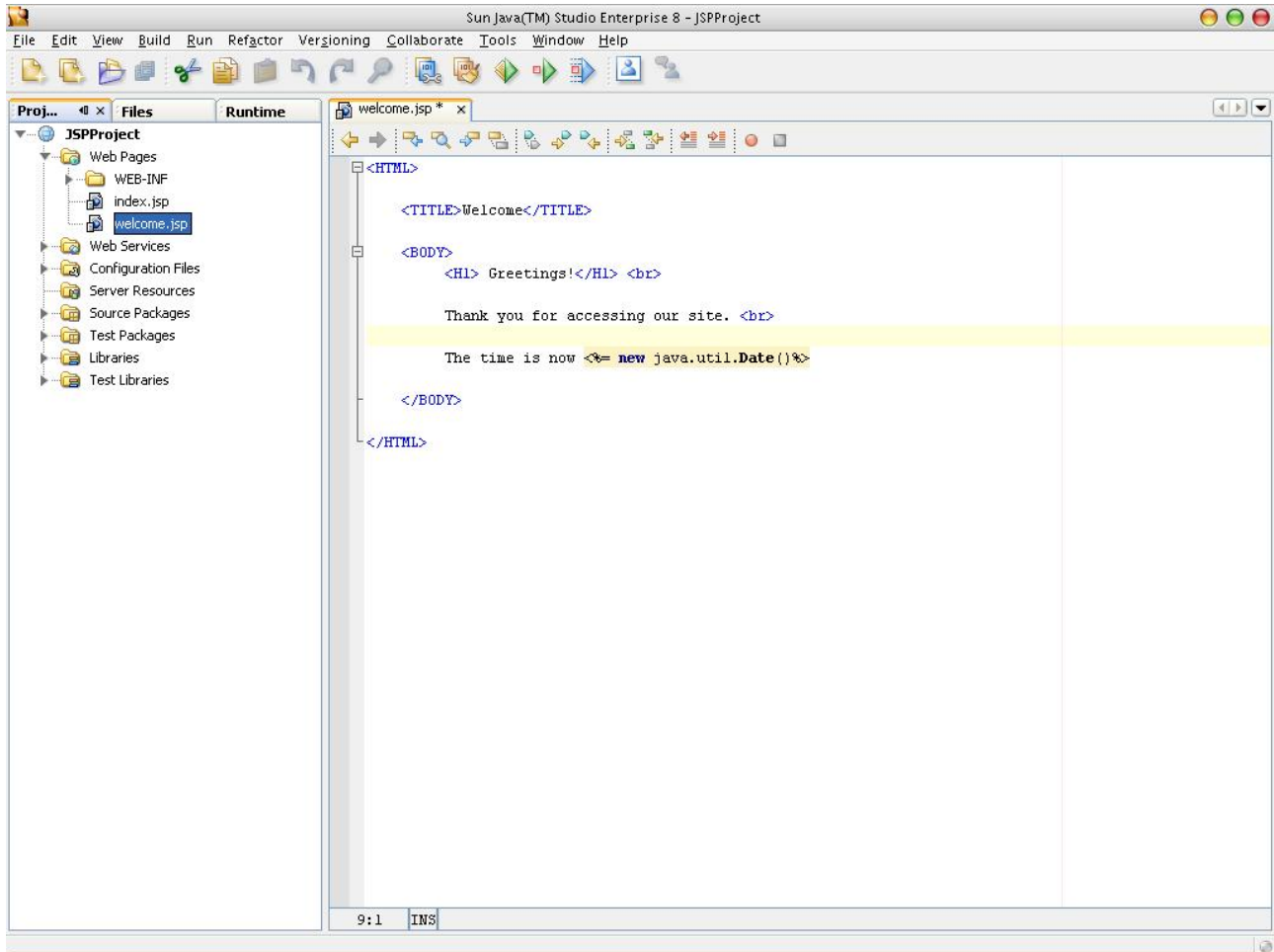
Hal ini merupakan bagian dari kode Java untuk menampilkan hari dan tanggal pada saat itu juga. Hal ini mempermudah untuk membuat objek baru untuk membuat objek Date dan menampilkannya sebagai String.

Dibawah ini merupakan hasil output dari file JSP tersebut diatas.



#### 4.2.4 Menggunakan IDE Enterprise

JSP dapat dijalankan dari berbagai macam projek aplikasi web pada IDE. Dengan asumsi bahwa projek telah siap.



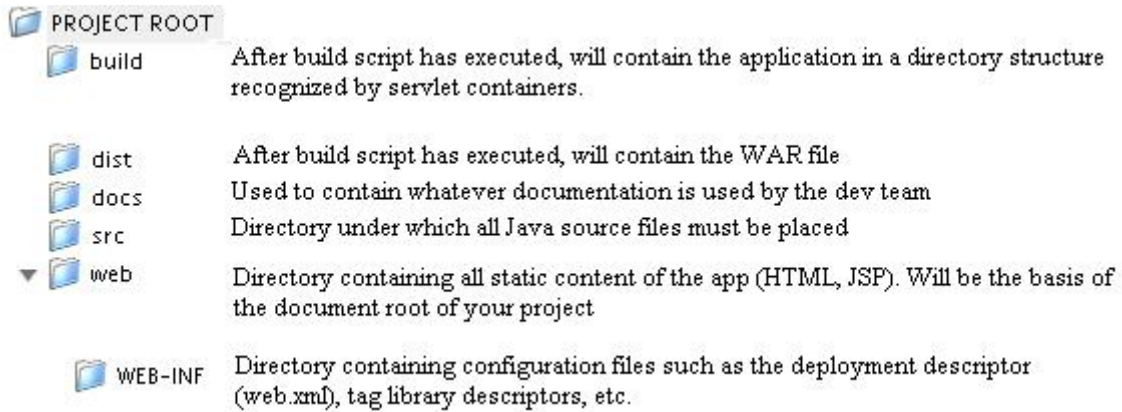
Spesifikasi dari file JSP ini kemudian dapat dijalankan dari IDE secara langsung dengan menekan SHIFT + F6. Alternatif lain, project web dapat dijalankan sebagai file WAR dan diupload kedalam server. Kemudian JSP dapat diakses dengan mengetikan URL berikut ini:

`http://[host]:[port]/[WEB_PROJECT_NAME]/[JSP_NAME]`

### 4.2.5 Menggunakan build tool

JSP juga dapat dijalankan dengan menyimpannya sebagai file WAR dengan menggunakan build tool (Seperti salah satu outlined pada bab Servlet Dasar), dan kemudian menjalankan file WAR kedalam web server.

Struktur direktori

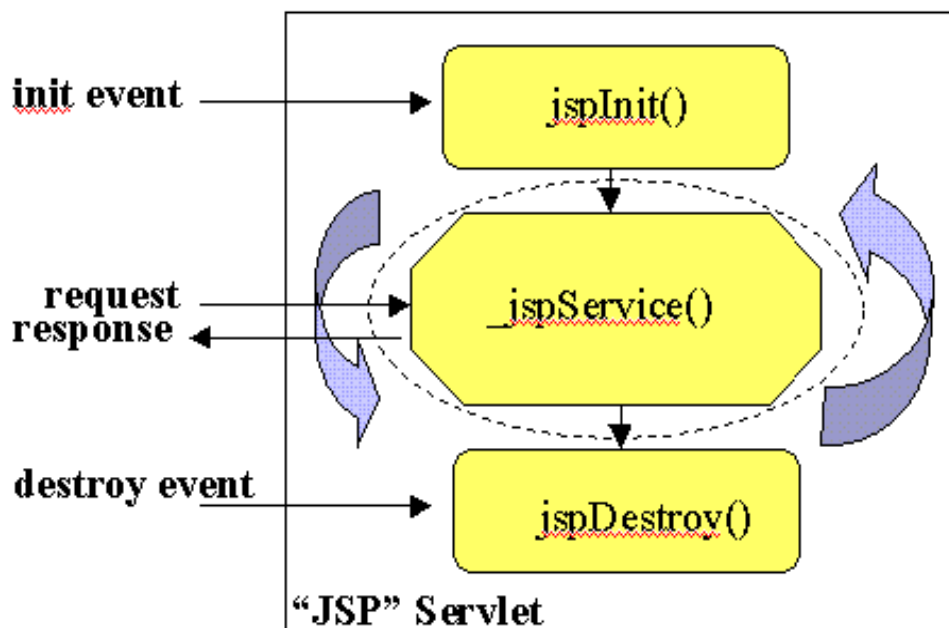


Pengembangan struktur direktori telah didiskusikan pada awal bab, file JSP harus disimpan pada web direktori.

### 4.2.6 Alur JSP

Container servlet mengatur JSPs pada suatu cara untuk mengatur servlet itu sendiri : Melalui penggunaan suatu alur JSP maka dapat dijalankan dengan baik.

JSPs memiliki tiga fase alur : inialisasi, servis, dan destruksi. Fase-fase ini sama dengan method servlet yang diambil dari container yang berbeda : `jspInit()` untuk inialisasi fase, `_jspService()` untuk servis fase, dan `jspDestroy()` untuk mendestruksi fase.



Dari contoh JSP yang telah diberikan, terlihat membingungkan untuk membahas method `jspInit` atau `_jspService()`. Contoh dari JSP hanya simple text page yang kebanyakan berasal dari content HTML : dia tidak memiliki method yang lain. Jawaban dari hal tersebut adalah : JSPs di-compile kedalam class servlet yang sama oleh server. Hal ini menyebabkan class servlet yang menangani permintaan untuk page JSP. Translasi ini dimasukkan kedalam servlet dan kompilasi subsequent telah selesai digunakan oleh server : para pengembang tidak perlu ragu-ragu tentang bagaimana prosedur ini diselesaikan.

Jika anda berharap untuk melihat class servlet yang di-generate, instalasi dari sun aplikasi server 8.1 menempatkan mereka pada direktori ini :

```
${SERVER_HOME}/domains/domain1/generated/jsp/j2ee-modules/  
${WEB_APP_NAME}/org/apache/jsp
```

Pada halaman selanjutnya merupakan persamaan servlet dari contoh JSP kita.

---

```
Package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private static java.util.Vector _jspx_dependants;

    public java.util.List getDependants() {
        return _jspx_dependants;
    }

    public void _jspService(HttpServletRequest request, HttpServletResponse
response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;

        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html");
            response.addHeader("X-Powered-By", "JSP/2.0");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("<HTML>\n");
            out.write("\n");
            out.write("    <TITLE>Welcome</TITLE>\n");
        }
    }
}
```

---

```

out.write("\n");
out.write("    <BODY>\n");
out.write("        <H1> Greetings!</H1> <br>\n");
out.write("\n");
out.write("        Thank you for accessing our site. <br>\n");
out.write("\n");
out.write("        The time is now ");
out.print( new java.util.Date());
out.write("\n");
out.write("\n");
out.write("    </BODY>\n");
out.write("\n");
out.write("</HTML>");
} catch (Throwable t) {
    if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
        if (_jspx_page_context != null)
            _jspx_page_context.handlePageException(t);
    }
} finally {
    if (_jspxFactory != null)
        _jspxFactory.releasePageContext(_jspx_page_context);
}
}
}

```

Bukan hal yang penting untuk mengerti kode diatas. Hal yang penting disini adalah untuk melihat bahwa JSPs ditangani seperti Servlet, meskipun tidak dapat dilihat secara jelas. Poin lain disini adalah bahwa JSP merupakan Servlet. Mereka hanya berbeda pada cara seorang pengembang dalam menghasilkan content : JSPs lebih berorientasi pada text, meskipun servlets memperbolehkan para pengembang menggunakan kode Java.

## 4.3 Syntax JSP dan Semantics

Meskipun JSP berbasis Java, dan dikendalikan sebagai kode Java oleh servlet, memperbolehkan pengembang untuk menggunakan syntax yang berbeda pada spesifikasi Java 2.0 dan sebagai gantinya menggunakan aturan spesifikasi JSP. Bagian berikut ini menggambarkan syntax JSP dengan lebih detail.

### 4.3.1 Element-element dan Data Template

Semua komponen Java Server Pages dapat dibagi menjadi dua kategori umum: *elements* dan *templates data*. Element merupakan dynamically yang menghasilkan informasi. Data template merupakan informasi static yang memperhatikan presentasi. Pada hello.jsp, ekspresi JSP, `<%= new java.util.Date() %>` adalah satu-satunya element data yang memanggil data template.

#### Listing 1: hello.jsp

```

<html>
<head>
<title>Hello World!</title>
</head>
<body>

```

```
<center>
<h1>Hello World! It's <%= new java.util.Date()%></h1>
</center>
</body>
</html>
```

### 4.3.2 Dua Tipe Syntax

Dua tipe dari authoring JSP didukung oleh Container JSP : JSP Style dan XML Style. Keduanya termasuk dalam bab ini. Memilih salah satu format syntax hanya bergantung dari preference dan standarisasi. Normal syntax didesain lebih mudah untuk pada pembuat(author). XML-compatible yntax telah disediakan ketika menggunakan JSP authoring tools. Bagaimanapun juga, yang lebih sering disediakan adalah normal syntax karena dia lebih mudah untuk dibaca dan dimengerti. Text ini akan menggunakan normal syntax pada contoh-contohnya.

### 4.3.3 Scripting Elements

Seperti yang telah dijelaskan pada bab sebelumnya, JSPs memungkinkan untuk dilihat sebagai HTML atau XML dokumen dengan berdasar pada Script JSP. Scripting JSP element memperbolehkan anda memasukkan kode Java kedalam Servlet yang akan di-generate dari halaman JSP.

Cara termudah untuk membuat dynamic JSP adalah dengan menaruh scripting element kedalam data template.

Pada bab ini Anda akan mempelajari scripting element JSP berikut ini:

1. Scriptlets,
2. Expressi, dan
3. Deklarasi.

### 4.3.4 Scriptlets ( <% ... %> )

Scriptlets menyediakan cara untuk memasukkan bits dari kode Java diantara chunks dari data template dan memiliki form berikut ini :

```
<% Java code; %>
```

Menggambarkan kode Java diantara <% dan %> sama seperti menulis kode Java secara normal kecuali disana tidak dibutuhkan untuk deklarasi class. Scriptlets bagus digunakan pada kode java seperti pernyataan kondisional loops, dll. Disana tidak ada batasan secara spesifik sebagai kompleksitas dari kode java yang harus disimpan diantara scriptlets. Bagaimanapun juga, kita harus melakukan tindakan pencegahan pada pemakaian scriptlets yang berlebihan. Menaruh perhitungan yang berat didalam scriptlets merupakan isu dari penggunaan kode. Juga, Menggunakan scriptlets extensively violates JSPs role menjadi komponen presentation layer yang utama.

Berikutnya kita akan mendiskusikan bagaimana kita dapat menggunakan JavaBeans untuk enkapsulasi data results passed dari komponen lain, Secara drastis mengurangi jumlah dari scriptlets yang dibutuhkan pada halaman. Bahkan kemudian, kita akan mendiskusikan bagaimana cara menggunakan custom tags untuk menyediakan common tasks seperti looping, logic branching, dll. Hal ini dikombinasikan dengan JavaBeans menggunakan cut scriptlet.

---

Jika anda ingin menggunakan karakter "%>" didalam scriptlet, tulislah "%\>" sebagai gantinya. Hal ini akan mencegah compiler dari penginterpretasian karakter sebagai penutup tag scriptlet.

Dibawah ini merupakan dua contoh yang menggambarkan contoh kode Java yang sangat simple diantara tag HTML.

### Contoh: Simple Println

PrintlnScriptlet.jsp

```
1 <html>
2 <head>
3 <title>Scriptlet Example 1</title>
4 </head>
5 <body>
6 <%
7 String username="jedi";
8 out.println ( username ) ;
9 %>
10 </body>
11 </html>
```

Contoh diatas merupakan ilustrasi yang mudah tentang kode Java didalam JSP Anda. Kode diatas menghasilkan output text, jedi, kedalam web browser.

### Contoh: For-Loop pada scriptlet

LoopScriptlet.jsp

```
1 <html>
2 <head>
3 <title>Scriptlet Example 2</title>
4 </head>
5 <body>
6 <% for (int i=0; i < 10; i++) { %>
7 This line is printed 10 times.
8 <br/>
9 <% } %>
10 </body>
11 </html>
```

Contoh diatas menampilkan implementasi kode Java dari for-loop didalam tag scriptlet(<%...%>).

Untuk outputnya dapat Anda lihat setelah menjalankan JSP ini dan menghasilkan text "This line is printed 10 times." Dicitak 10 kali dalam kaitannya dengan menjalankan for-loop untuk iterasi 0 sampai 9 seperti yang ditampilkan pada ilustrasi 8.1.

---

Ingatla bahwa scriptlet itu sendiri tidak dikirim ke client tetapi hanya sebagai outputnya. Cobalah untuk melihat source dari output JSP yang anda hasilkan di browser Anda untuk memahami point ini. Anda harus melihat tag-tag HTML dan output scriptlet.

Akhirnya, syntax XML-compatible untuk menulis scriptlets adalah:

```
<jsp:declaration>
Java code;
</jsp:declaration>.
```

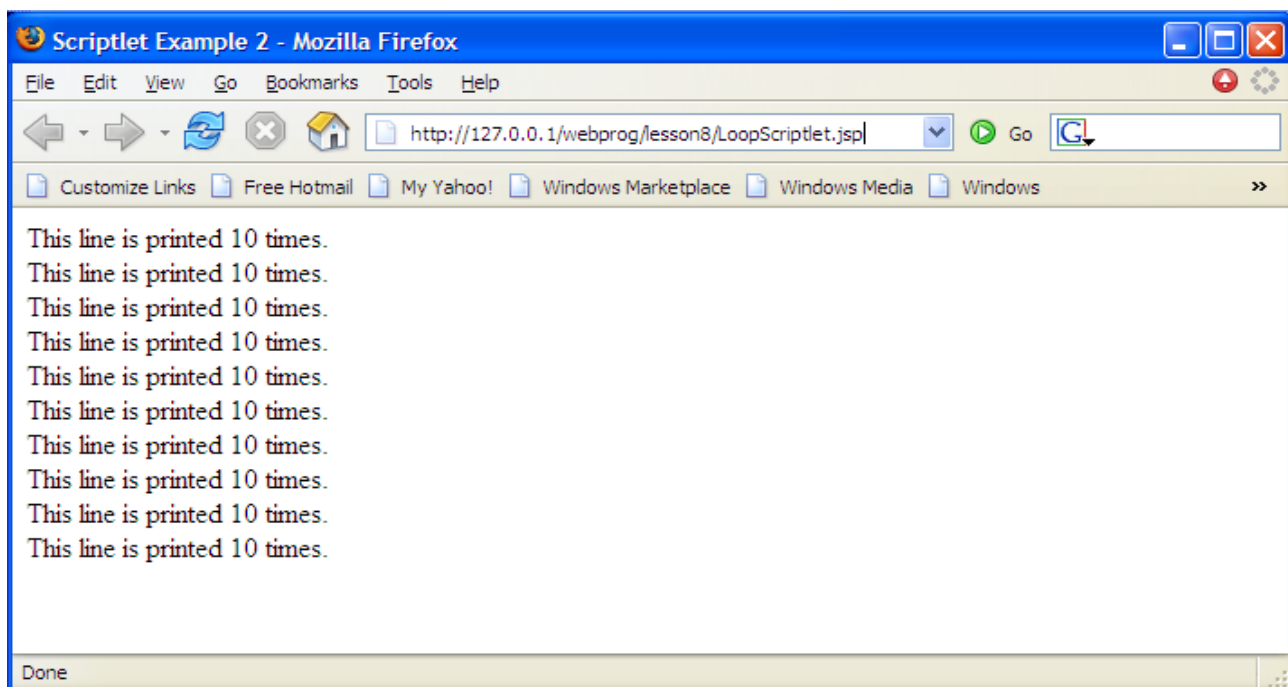
#### 4.3.5 Expressi ( <%= %> )

Expressi menyediakan cara untuk memasukkan nilai Java secara langsung kedalam output. Hal ini memiliki form seperti berikut ini:

```
<%= Java Expression %>
```

Sebenarnya hal ini dapat menggunakan out.println().

Catatan bahwa tanda ( ; ) tidak ditampilkan pada akhir kode didalam tag.



Ilustrasi 8. 1: LoopScriptlet.jsp Output

Expressi java yang lain disimpan diantara <%= dan %> yang dievaluasi pada saat run-time, yang dikonversi kedalam string, dan dimasukkan kedalam page. Expressi selalu mengirim string text ke client, tetapi object yang dihasilkan sebagai hasil dari expressi yang tidak memerlukan nilai instan object sebagai string. Semua yang bukan object string secara instan dikonversi melalui method toString(). Jika hasilnya primitive, kemudian string primitive akan ditampilkan.

Hal tersebut akan dijelaskan terlebih dulu bahwa pada saat run-time(ketika ada request page), hal ini akan memberikan expressi akses penuh untuk informasi tentang request.

Sebuah nomer dari variabel yang telah dikenali sebenarnya telah tersedia untuk author JSP untuk expressi yang mudah. Variable yang telah dikenali ini disebut **implicit objects** dan berikut ini akan kita bahas secara detail. Untuk tujuan dari expression, yang terpenting

adalah:

- o `request, HttpServletRequest;`
- o `response, HttpServletResponse;`
- o `session, HttpSession` associated dengan request (jika ada); dan
- o `out, PrintWriter` (versi buffered dari tipe `JspWriter`) digunakan untuk mengirim output ke client.

Sebagai contohnya, untuk mencetak hostname, Anda hanya butuh untuk memasukkan expressi jsp dibawah ini:

```
Hostname: <%= request.getRemoteHost() %>
```

Akhirnya, syntax XML-compatible syntax untuk `<%= Java Expression %>` adalah:

```
<jsp:expression>  
Java Expression  
</jsp:expression>
```

#### 4.3.6 Deklarasi ( `<%! %>` )

Deklarasi memperbolehkan anda untuk menggambarkan method atau variable. Dia memiliki form seperti dibawah ini:

```
<%! Java Code %>
```

Deklarasi digunakan untuk meletakkan kode hanya seperti scriptlet tetapi deklarasi dapat dimasukkan kedalam main body dari class servlet, diluar dari proses request method `_jspService()`. Untuk alasan ini kode diletakkan pada deklarasi dapat digunakan untuk mendeklarasikan method baru dan variabel class global. Dalam hal yang lain, kode pada deklarasi tidak selalu aman, kecuali diprogram terlebih dahulu oleh author JSP, perlu diperhatikan adalah padasaat menulis deklarasi JSP.

Berikut ini merupakan hal yang perlu diingat dalam menggunakan tag deklarasi:

- o Sebelum deklarasi Anda harus menuliskan `<%!`
- o Pada akhir deklarasi, developer harus menuliskan `%>`
- o Kode pada tag harus diakhiri dengan tanda baca `( ; )`.
- o Deklarasi tidak men-generate outptt tetapi digunakan dengan expressi JSP atau scriptlet.

Sejak deklarasi tidak men-generate outptu apapun, mereka sering digunakan pada konjungsi dengan expressi JSP atau scriptlet. Sebagai contohnya, ada sebuah JSP yang mencetak angka dari waktu current page telah direquest sejak server atau class servlet telah dirubah.

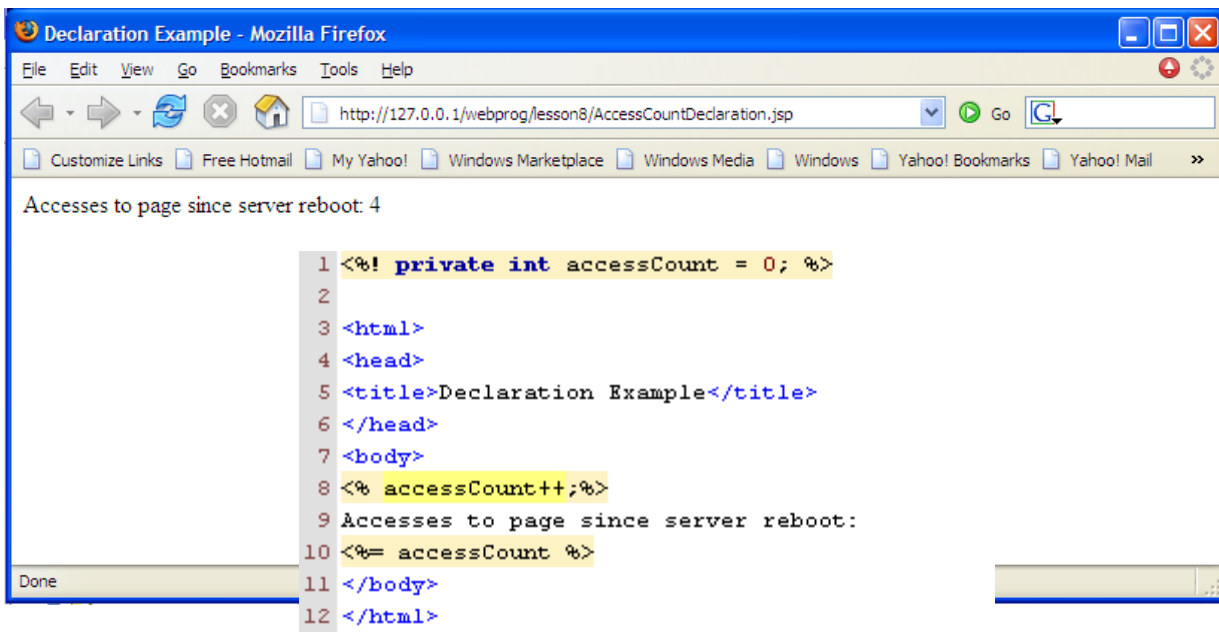
**Contoh**

## AccessCountDeclaration.jsp

JSP dapat mencetak angka dari visits dengan mendeklarasikan variabel class-wide `accessCount`, menggunakan scriptlet untuk meningkatkan nilai dari angka dari sebuah page visit dan sebuah expressi untk menampilkan nilainya.

Illustrasi 8.3 menampilkan contoh dari JSP page yang direfresh sebanyak 4 kali.

*Illustrasi 8.2: AccessCountDeclaration.jsp output*



Untuk mengetahui bagaimana JSP page ditranslasi kedalam sebuah servlet mari kita uji output servlet dari container JSP kita untuk `AccessCountDeclaration.jsp`. Container JSP men-generate sebuah file Java yang memanggil `AccessCountDeclaration_jsp.java` dan isinya. Ingatlah bahwa deklarasi, scriptlet dan expressi telah digaribawahi untuk referensi termudah.

**Listing 2: AccessCountDeclaration\_jsp.java**

```
package org.apache.jsp.JSP;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class AccessCountDeclaration_jsp
    extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

    private int accessCount = 0;

    private static java.util.Vector _jspx_dependants;

    public java.util.List getDependants() {

        return _jspx_dependants;
    }
}
```

```
}

public void _jspService (
    HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;

    try {

        _jspxFactory = JspFactory.getDefaultFactory();
        response.setContentType("text/html");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);

        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();

        out = pageContext.getOut();
        _jspx_out = out;
        out.write("\n");
        out.write("\n");
        out.write("<html>\n");
        out.write("<head>\n");
        out.write("<title>Declaration Example</title>\n");
        out.write("</head>\n");
        out.write("<body>\n");
        accessCount++;
        out.write("Accesses to page since server reboot: \n");
        out.print( accessCount );
        out.write("\n");
        out.write("</body>\n");
        out.write("</html>\n");
        out.write("\n");
        out.write("\n");

    } catch (Throwable t) {

        if (!(t instanceof SkipPageException)){

            out = _jspx_out;
            if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();

            if (_jspx_page_context != null)
                _jspx_page_context.handlePageException(t);

        }

    } finally {

        if (_jspxFactory != null)
            _jspxFactory.releasePageContext(_jspx_page_context);

    }

}
```

```
    }  
}
```

Ingatlah bagaimana deklarasi untuk `accessCount` diletakkan diluar method `_jspService()` sebagai anggota variable. Hal ini membuat `accessCount` diperbolehkan tidak hanya untuk method `_jspService()` tetapi untuk method yang lain digambarkan dalam JSP. Menguji kode terdahulu menunjukkan kepada kita tentang penempatan yang tepat sebuah deklarasi, scriptlets dan expressi pada kode source Java yang ditranslasi dari page HTML.

Akhirnya, ingatlah bahwa syntax the XML-compatible syntax untuk `<%! JavaCode %>` adalah

```
<jsp:declaration>  
Java Code;  
</jsp:declaration>
```

### Template Text

- o Gunakan `<\%` untuk menghasilkan `<%` pada output
- o `<%-- JSP Comment --%>`
- o `<!-- HTML Comment -->`
- o Seluruh text di luar JSP ditampilkan pada halaman.

### 4.3.7 Variable-variable yang telah dikenal

Pada diskusi tentang tag expressi kita menemukan object yang tersembunyi. Pembahasan ini menggambarkan object tersebut secara detail.

Object-object JSP yang tersembunyi secara otomatis dideklarasikan oleh container JSP dan selalu tersedia dalam penggunaan expressi dan scriptlets (tetapi tidak didalam deklarasi). Berikut ini merupakan daftar object-object yang tersembunyi (implicit objects):

**request:** Kejadian dari object `javax.servlet.http.HttpServletRequest` dihubungkan dengan request dari client.

**response:** Kejadian dari object `javax.servlet.http.HttpServletResponse` dihubungkan dengan response ke client.

**pageContext:** Object `PageContext` dihubungkan dengan halaman sebelumnya.

**out:** Referensi object `javax.servlet.jsp.JspWriter` dapat digunakan untuk menulis action dan data template dalam page JSP, sama dengan hal tersebut dari object `PrintWriter` kita gunakan pada saat mendiskusikan servlet. Variable tersembunyi (implicit variable) `out` secara otomatis diinialisasi dengan menggunakan method pada object `PageContext`.

**session:** Kejadian dari object `javax.servlet.http.HttpSession`. Kejadian ini sama untuk memanggil method `HttpServletRequest.getSession()`.

**application:** Object `ServletContext` is an instance of the `javax.servlet.ServletContext`. Kejadian ini sama dengan memanggil method `getServletConfig().getContext()`. Implicit object ini disharing oleh semua servlets dan JSP pages pada server.

**config:** `Config` dari implicit object merupakan kejadian dari object `javax.servlet.ServletConfig` untuk page ini. Sama halnya dengan sebagai Servlets, JSPs memiliki akses untuk menginisialisasi parameter yang terdapat pada Web Application Deployment Descriptor.

### 4.3.8 JSP Directives

Directive merupakan pesan untuk container JSP. Mereka mempengaruhi seluruh struktur dari class servlet. Hal tersebut memiliki form seperti berikut ini:

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

Daftar untuk mengeset atribut juga dapat disebutkan satu persatu untuk single directive seperti dibawah ini:

```
<%@ directive attribute1="value1"
      attribute2="value2"
      ...
      attributeN="valueN" %>
```

**Catatan:** Whitespaces setelah <%@ dan sebelum %> merupakan pilihan/opsional.

Directive tidak menampilkan berbagai hasil output ketika ada permintaan pada page tetapi mereka merubah cara JSP engine dalam memproses page tersebut. Sebagai contohnya, Anda dapat membuat session data yang tidak tersedia pada page dengan mengeset directive page (session) bernilai false.

JSP directive memberikan informasi yang spesial tentang page kepada JSP Engine. Directive dapat menjadi satu dalam page, yaitu include atau taglib. Masing-masing directives tersebut memiliki setting atributnya sendiri-sendiri.

#### 4.3.8.1 Page directive

Page directive menggambarkan proses informasi untuk sebuah page. Hal tersebut memperbolehkan Anda untuk meng-import class, customize superclass servlet.

Directive memiliki pilihan atribut seperti dibawah ini yang menyediakan JSP Engine dengan proses informasi yang spesial. Ingatlah bahwa dibawah ini adalah daftar atribut yang bersifat cesa-sensitive:

<i>Directive</i>	<i>Pengertian</i>	<i>Contoh Penggunaan</i>
<b>extends</b>	Superclass digunakan oleh JSP Engine untuk men-translasi Servlet. Menjadi translator untuk keyword extends pada bahasa pemrograman Java.	<%@ page extends = "com.taglib..." %>
<b>Language</b>	Mengindikasikan scripting language yang digunakan pada scriptlets, expressi dan deklarasi yang ditemukan pada page JSP yang digunakan. Untuk atribut ini hanya menggambarkan nilai java.	<%@ page language = "java" %>
<b>import</b>	Mengimport class pada package java kedalam halaman JSP.	<%@ page import = "java.util.*" %>

<i>Directive</i>	<i>Pengertian</i>	<i>Contoh Penggunaan</i>
<b>Session</b>	Menandai apakah page menggunakan sessions. Untuk session data pada semua page JSP memiliki default session data available. Merubah session ke false memiliki hasil yang benefit.	Defaultnya adalah true.
<b>buffer</b>	Mengontrol penggunaan output buffered untuk page JSP. Jika bernilai none, maka tidak ada buffer dan output ditulid secara langsung disesuaikan pada <code>PrintWriter</code> . Nilai default dari buffer adalah 8kb.	<code>&lt;%@ page buffer = "none" %&gt;</code>
<b>autoFlush</b>	Ketika diset true, flushes akan aktif ketika output buffer penuh.	<code>&lt;%@ page autoFlush = "true" %&gt;</code>
<b>isThreadSafe</b>	Mengindikasikan jika Servlet yang di-generate setuju dengan berbagai request. Jika bernilai true, thread yang baru akan menangani request secara bersamaan. Nilai defaultnya adalah true.	
<b>info</b>	Para pengembang menggunakan info atribut untuk menambah informasi /dokumen untuk suatu page. Biasanya digunakan untuk menambah author, versi, copyright dan info tanggal.	<code>&lt;%@ page info = "jeni.org test page, alpha version " %&gt;</code>
<b>errorPage</b>	Menampilkan halaman yang berisi tentang kesalahan yang terjadi. URL harus menuju ke halaman error.	<code>&lt;%@ page errorPage = "/errors.jsp" %&gt;</code>
<b>IsErrorPage</b>	Sebuah flas harus diset true untuk membuat halaman JSP menjadi halaman error. Halaman ini memiliki akses untuk implicit object exception.	

Syntax XML untuk menggambarkan directives adalah  
`<jsp:directive.directiveType attribute=value />`

Contoh, persamaan dari XML

```
<%@ page import="java.util.*" %>
adalah
<jsp:directive.page import="java.util.*" />
```

#### 4.3.8.2 Include directive

Include directive menggambarkan file-file yang harus dimasukkan pada halaman. Hal tersebut membolehkan Anda memasukkan sebuah file kedalam class Servlet ( untuk memasukkan isi dari sebuah file yang berada didalam file lain) selama **translation time**. Biasanya, include file digunakan untuk navigasi, table, header dan footer, dll. – komponen-komponen yang diperbolehkan untuk berbagai halaman.

Berikut ini syntax untuk include directive:

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

Contoh, jika Anda ingin untuk memasukkan sebuah menubar yang akan ditemukan pada directory yang Anda inginkan, Anda dapat menulis:

```
<%@ include file = "menubar.jsp" %>
```

#### 4.3.8.3 Tag Lib directive

Sebuah tag library merupakan kumpulan dari custom tags. Taglib directive menggambarkan bagaimana tag library digunakan pada sebuah halaman. Berikut ini cara penulisan taglibs:

```
<%@ taglib uri = "tag library URI" prefix = "tag Prefix" %>
```

Directive ini memberitahu container dimana markup pada halaman harus disesuaikan dengan dengan custom code dan kode markup link apa yang dipakai. Kita ambil contohnya dari index.jsp pada listing dibawah ini. Pada baris pertama menjelaskan bahwa index.jsp menggunakan custom code "struts/template" dan prefis "template" untuk penulisan yang lebih mudah. Penggantian referensi taglib yang baik dilakukan dengan memrefix markup-nya.

#### index.jsp

```
<%@ taglib uri="struts/template" prefix="template" %>

<template:insert template="/WEB-INF/view/template.jsp">
  <template:put name="header" content="/WEB-INF/view/header.jsp" />
  <template:put name="content" content="/WEB-INF/view/index_content.jsp" />
  <template:put name="footer" content="/WEB-INF/view/footer.jsp" />
</template:insert>
```

Custom tag yang telah dikenalkan pada JSP 1.1 dan memperbolehkan para pengembang JSP untuk menyembunyikan complex server side code dari web designer.

## 4.4 JavaBeans pada JSP

Kegunaan dari JavaBeans tidak hanya digunakan pada spesifikasi JSP. Bagaimanapun juga, mereka menyediakan fungsi yang mudah digunakan, oleh karena itu kegunaan dari JavaBean dapat sangat mengurangi jumlah dari elemen scripting yang dapat ditemukan pada sebuah halaman Java.

Yang paling utama, dari latihan JavaBean adalah: JavaBean hanya merupakan class Java sederhana yang mempertahankan standart peng-codingan:

- Menyediakan default, konstruktor tanpa argumen
- Menyediakan method get dan set untuk properti yang akan digunakan.

Berikut ini contoh dari class JavaBean yang sederhana:

```
package jeni.beans;

public class User {
    private String name;
    private String address;
    private String contactNo;

    public User() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getContactNo() {
        return contactNo;
    }

    public void setContactNo(String contactNo) {
        this.contactNo = contactNo;
    }
}
```

Bagaimana seharusnya JavaBean digunakan dalam JSP?

- Sebagai Data Transfer Object - JavaBeans lebih sering digunakan dalam JSPs sebagai object yang mengirim data dari source lain. Pada sebagian besar aplikasi, proses diselesaikan didalam servlet, bukan didalam JSP. Pada suatu form dari satu atau lebih JavaBean tidak hanya sebuah hasil yang dilewatkan kedalam JSP.
- Sebagai Helper object - pada beberapa aplikasi kecil, tidak efisien untuk memiliki separate servlet untuk memproses data. Pada kasus ini, lebih baiknya menyimpan fungsi-fungsi kedalam JavaBean dimana dapat diakses oleh JSP.

#### 4.4.1 JavaBeans-dihubungkan dengan JSP Action

JSP menggambarkan bahwa standard action menyederhanakan kegunaan dari JavaBean.

##### <jsp:useBean>

Untuk menggunakan komponen Java Bean, hal pertama yang harus dipikirkan adalah memungkinkan penggunaan sebuah bean dengan halaman melalui *instantiation*, apa yang dapat kita lakukan dengan action tersebut.

Berikut ini adalah attribute dari <jsp:useBean> action:

- **id** – Atribut ini menetapkan nama dari bean dan bagaimana Anda dapat menunjuk bean pada sebuah halaman.
- **scope** – Atribut ini menetapkan lingkup dimana Anda akan menyimpan bean instance. Scope dapat diset ke page(sebagai default), session, request, atau application.
- **class** - Atribut ini menetapkan class Java dari bean berasal. Jika Anda menetapkan namaBean, Anda tidak perlu menetapkan classnya.
- **beanName** - Atribut ini menetapkan nama dari bean yang disimpan pada server. Anda dapat menunjuk kepadanya dengan sebuah class (contoh, com.projectalpha.PowerBean). Jika Anda menetapkan class, Anda tidak perlu untuk menetapkan beanName.
- **type** - Atribut ini menetapkan tipe dari variabel scripting yang dikembalikan oleh bean. Tipe tersebut harus dihubungkan ke class dari bean.

Berikut ini merupakan contoh bagaimana menggunakan Usesr JavaBean pada halaman JSP:

```
<jsp:useBean id="user" scope="session" class="jeni.bean.User"/>
```

Ketikan halaman menghadapi useBean action, pertama perlu mencoba untuk melihat jika disana telah ada JavaBean instance yang memberikan lokasi tipe dalam scope yang disediakan. Jika ada, maka halaman akan membuat use dari bean instance. Jika tidak ada, secara otomatis container akan membuat bean instance yang baru menggunakan default bean constructor tanpa argument dan meletakkan bean pada scope yang disediakan.

Jika kemampuan diatas digambarkan sebagai scriptlet, akan terlihat seperti berikut ini:

```
<%
  jeni.bean.User user = (jeni.bean.User)session.getAttribute("user");
  if (user == null) {
    user = new User();
    session.setAttribute("user", user);
  }
%>
```

##### <jsp:getProperty>

Action ini mendapatkan kembali nilai dari properti yang ditetapkan didalam JavaBean dan seketika menghasilkan output sebagai tanggapan.

Action ini memiliki dua atribut :

- **name** – nama dari JavaBean yang propertinya akan didapatkan kembali. Hal ini harus memiliki nilai yang sama sebagai id atribut yang digunakan pada awal <jsp:useBean> action
- **property** – nama dari properti yang valuenya akan didapatkan kembali.

Jika para pengembang berharap untuk dapat mendapatkan kembali nilai dari sebuah properti JavaBean tanpa menghasilkan output sebagai tanggapan, maka tidak ada pilihan jika

menggunakan scriptlets atau EL (akan dibahas pada bab berikutnya).

Melanjutkan dari latihan kita sebelumnya, untuk menampilkan nama properti dari the User JavaBean, kita dapat menggunakan the getProperty action dibawah ini:

```
<jsp:getProperty name="user" property="name"/>
```

### <jsp:setProperty>

Action ini memperbolehkan para pengembang untuk mengeset properti yang diberikan JavaBean tanpa harus menulis kode scriptlet.

Action ini memiliki properti yang sama dengan getProperty action, dengan beberapa penambahan:

- value – sebuah nilai harus diset didalam properti. Hal ini dapat berupa nilai static atau sebuah ekspresi yang dapat dievaluasi pada saat runtime.
- param – menetapkan parameter request dari properti yang akan mendapatkan kembali sebuah nilai. Penempatan \* sebagai nilai param

Para pengembang menggunakan action ini harus dengan satu spesifikasi dari atribut value atau param. Pada kedua atribut tersebut didalam action akan menyebabkan suatu pengecualian/exception yang harus dihilangkan.

## 4.5 Penanganan Error

JSPs dapat menggunakan halaman yang directive untuk menspesifikasi halaman yang akan menangani berbagai exception. Atribut errorPage dari page directive dapat dilewatkan sebuah relatif URL ke JSP page yang didesain untuk halaman yang error. Halaman yang didesain untuk error atribut isErrorPage dapat diset true. Hal ini akan memberikan akses untuk implicit object yang dinamai exception – hal ini berisi tentang detail dari exception.

Berikut ini adalah contoh untuk errorPage :

```
<% @page isErrorPage="true"%>
<HTML>
  <TITLE>Error!</TITLE>
  <BODY>
    <H1>An Error has occurred.</H1><br/>
    Sorry, but an error has occurred with the page you were previously accessing. Please contact any member of the
    support team, and inform them that <%= exception.getMessage() %> was the cause of the error
  </BODY>
</HTML>
```