JSP2 Expression Language (EL)

- Avantajele folosirii EL
- Sintaxa de baza Expression Language
- Referirea variabilelor de scop
- Accesarea proprietatilor unui bean
- Accesarea variabilelor ne-scalare: array, list, map
- Operatorii EL
- Evaluarea conditionala a expresiilor

1. Beneficiile JSP2 Expression Language (EL).

Fiecare din tehnologiile de tip servlet si a paginilor JSP folosite in dezvoltarea unei aplicatii web prezinta atat avantaje cat si dezavantaje. Astfel, folosirea paginilor JSP are urmatoarele avantaje:

- Utila pentru dezvoltarea si mentenanta codului HTML
- Se pot include sectiuni de cod simplu in scripleti
- Pentru aplicatii mai complexe se folosesc clase utilitare apelate din sectiunile de scripting
- Se pot folosi java bean-urile si tagurile custom

Dezavantajele paginilor JSP se manifesta in aplicatiile web ce contin procesari complexe. De asemenea, aplicatia devine *page-based* ceea ce ofera un aspect simplist acesteia.

O aplicatie web construita cu servleti functioneaza bine cand iesirea este binara (ex: imagini), cand exista multiple procesari, apeluri, redirectari sau cand layout-ul paginii este variabil.

Combinarea paginilor JSP cu servletii este utila pentru aplicatii complexe care necesita atat procesari de date cat si layout-uri diferite de prezentare. De asemenea, echipa de dezvoltare este mare atat pentru partea de web cat si pentru partea de business logic. Combinarea celor doua tehnologii, servleti cu JSP, se numeste model view controller (MVC – Model View Controller) sau Model 2.

Ca si optiuni de implementare exista doua solutii:

- Folosirea RequestDispatcher pentru integrare JSP si servleti (aplicatii moderat complexe)
- Folosirea framework-urilor predefinite: Struts, Java Server Faces (JSF), Spring, Enterprise Java Beans (EJB)

Etapele de implementare a unei aplicatii web pe o arhitectura MVC sunt urmatoarele:

1. Definirea bean-urilor pentru reprezentarea datelor introduce de utilizatori

2. Se foloseste un servlet pentru a gestiona cererile (“requests”) venite din formularele de introducere date. Acest servlet citeste si valideaza parametrii transmisi, apeleaza metode de business logic.

3. Populatea bean-urilor:
   - Servletul apeleaza cod business logic sau cod de acces la date;
   - Rezultatele sunt plasate in bean-urile definite la pasul 1
4. Stocarea bean-urilor la nivelul de cerere (request), sesiune sau context al servletului.
5. Se paseaza (*'forward')* cererea la pagina JSP:
   - Servletul determina care pagina JSP este necesara pentru a transfera controlul
6. Extragerea datelor din bean-uri:
   - jsp:useBean si jsp:getProperty
   - Nu creaza sau modifica bean-urile, doar extrage datele continute

Principalele dezavantaje ale modelului MVC sunt in pasul final de afisarea datelor din bean in pagina JSP:

- Tag-urile jsp:useBean si jsp:getProperty nu permit accesarea subproprietatilor din bean si necesita cod in pagina JSP
- Elementele de scripting in pagina JSP sunt greu de mentinut intr-o aplicatie mare dezvoltata de mai multi programatori

Pagina JSP de afisare date *SomePage.jsp*:

<jsp:useBean id="customer" type="somePackage.Customer" scope="request" />

<jsp:getProperty name="customer" property="name"/>

Scopul este de a folosi in pagina JSP de afisare o sintaxa mai simpla si concisa pentru a afisa valorile scalare sau ne-scalare a proprietatilor si subproprietatilor din bean-ul asociat.

Codul in pagina JSP devine:

<h1> Numele clientului este: ***${customer.name}****</h1>*

Avantajele folosirii expresiilor din JSP2 Expression Language:

- In modelul MVC se modifica prin simplificare doar paginile JSP de afisare, restul componentelor ramanand neschimbate (bean-uri, servleti, business logic).
- Acces concis la obiectele stocate cu *setAttribute* in contextul curent (*PageContext, HttpServletRequest, HttpSession, ServletContext);* ex: *${varName}*
- Accesarea simpla a proprietatilor si sub-proprietatilor unui bean; ex: *${company.companyName}; ${company.president.firstName}*
- Accesarea simpla a componentelor unei colectii (array, list, map); *${variable[indexOrKey]}*
- Accesarea simpla a datelor din cerere: parametri, cookie-uri etc

- Folosirea operatorilor aritmetici, relationali, logici
- Expresii conditionale: *${test?option1 : option2}*
- Conversii automate de tipuri
- Valori vide in loc de mesaje de eroare: in majoritatea cazurilor valorile lipsa sau NullPointerExceptions sunt gestionate ca siruri vide.

2. Sintaxa de baza Expression Language.

Expression Language este compatibila cu versiunea minima JSP 2.0 (minim servlet 2.4), implementat in containerul Apache Tomcat versiunea minim 5. Daca se foloseste fisierul de configurare *web.xml*, atunci el trebuie configurat astfel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation=
          "http://java.sun.com/xml/ns/j2ee web-app_2_4.xsd"
          version="2.4">
…
</web-app>
```

Elementele EL pot apare in cadrul unei pagini JSP atat in cadrul textului normal cat si in tag-urile JSP specifice. Sintaxa de baza este urmatoarea:

${expresie}

Exemplu:

<ul>

* Nume: ${expresie1}
*

* Adresa: ${expresie2}
*

*
*

<jsp:include page=”${expresie3}textNormal${expresie4}”/>

2.1 Accesarea variabilelor din “scope-ul” curent

Cautarea variabilelor in scope-ul curent se face in urmatoarea ordine: HttpServletRequest, HttpSessionRequest, ServletContext. Exemplu de afisarea variabilelor din scope-ul curent:

ScopedVarsServlet.java

```
@WebServlet("/scoped-vars")
public class ScopedVarsServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    request.setAttribute("attribute1", "First Value");
    HttpSession session = request.getSession();
    session.setAttribute("attribute2", "Second Value");
    ServletContext application = getServletContext();
    application.setAttribute("attribute3",
                             new java.util.Date());
    request.setAttribute("repeated", "Request");
    session.setAttribute("repeated", "Session");
    application.setAttribute("repeated", "ServletContext");
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/WEB-INF/results/scoped-vars.jsp");
    dispatcher.forward(request, response);
  }
}
```

scoped-vars.jsp

```
<HTML>
<HEAD><TITLE>Accessing Scoped Variables</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Accessing Scoped Variables
</TABLE>
<P>
<UL>
  <LI><B>attribute1:</B> ${attribute1}
  <LI><B>attribute2:</B> ${attribute2}
  <LI><B>attribute3:</B> ${attribute3}
  <LI><B>Source of "repeated" attribute:</B> ${repeated}
</UL>
   </BODY></HTML>
```

2.2 Accesarea proprietatilor unui bean

Folosind EL se pot accesa proprietatile si sub-proprietatile unui bean intr-o forma simpla si concisa:

${bean.numeProprietate}

${bean.numeProprietate.numeSubproprietate}

Exemplu de accesarea bean-urilor din EL:

BeanPropsServlet.java

```
@WebServlet("/bean-properties")
public class BeanPropsServlet extends HttpServlet {
	@Override
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

		Instructor persoana = new Instructor("Aflori", "Cristian");
		Disciplina obiect = new Disciplina("LTW",
				"Limbaje si Tehnologii Web");
		Curs curs = new Curs(persoana, obiect);
		request.setAttribute("curs", curs);
		RequestDispatcher dispatcher = request
				.getRequestDispatcher("/WEB-INF/results/bean-properties.jsp");
		dispatcher.forward(request, response);
	}
}
```

Curs.java

```
public class Curs {

	private Instructor instructor;
	private Disciplina disciplina;

	public Curs(Instructor pers, Disciplina disc) {
		instructor = pers;
		disciplina = disc;
	}
	public Instructor getInstructor() {
		return instructor;
	}
	public void setInstructor(Instructor instructor) {
		this.instructor = instructor;
	}
	public Disciplina getDisciplina() {
		return disciplina;
	}
	public void setDisciplina(Disciplina disciplina) {
		this.disciplina = disciplina;
	}
   }
```

Bean-properties.jsp

```
<HTML>
<HEAD><TITLE>Accessing Bean Properties</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
```

```
  Accessing Bean Properties
</TABLE>
<P>
<UL>
  <LI><B>Prenume instructor:</B> ${curs.instructor.prenume}
  <LI><B>Nume instructor:</B> ${curs.instructor.nume}
  <LI><B>Cod curs:</B> ${curs.disciplina.cod}
  <LI><B>Denumire curs:</B> ${curs.disciplina.denumire}
</UL>
    </BODY></HTML>
```

2.3 Accesarea colectiilor in EL

Accesarea colectiilor se face astfel:

${attributeName[entryName]}

Exemplu de utilizare colectii:

CollectionsServlet.java

```
@WebServlet("/collections")
public class CollectionsServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String[] firstNames = { "Bill", "Scott", "Larry" };
    List<String> lastNames = new ArrayList<String>();
    lastNames.add("Gates");
    lastNames.add("McNealy");
    lastNames.add("Ellison");
    Map<String,String> companyNames =
      new HashMap<String,String>();
    companyNames.put("Ellison", "Oracle");
    companyNames.put("Gates", "Microsoft");
    companyNames.put("McNealy", "Sun");
    request.setAttribute("first", firstNames);
    request.setAttribute("last", lastNames);
    request.setAttribute("company", companyNames);
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/WEB-INF/results/collections.jsp");
    dispatcher.forward(request, response);
  }
}
```

Collections.jsp

```
<HTML>
<HEAD><TITLE>Accessing Collections</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
```

```
  <TR><TH CLASS="TITLE">
  Accessing Collections
</TABLE>
<P>
<UL>
  <LI>${first[0]} ${last[0]} (${company["Gates"]})
  <LI>${first[1]} ${last[1]} (${company["McNealy"]})
  <LI>${first[2]} ${last[2]} (${company["Ellison"]})
</UL>
   </BODY></HTML>
```

2.4 Apelarea obiectelor predefinite si operatori in EL

Apelarea obiectelor predefinite:

- pageContext: *${pageContext.session.id}*
- param si paramValues*: ${param.custId}*
- header si headerValues: *${header.Accept}, ${header[“Accept”]}*
- cookie: *$cookie.userCookie.value*

implicit-objects.jsp

```
<HTML>
<HEAD><TITLE>Using Implicit Objects</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Using Implicit Objects
</TABLE>
<P>
<UL>
  <LI><B>test Request Parameter:</B> ${param.test}
  <LI><B>User-Agent Header:</B> ${header["User-Agent"]}
  <LI><B>JSESSIONID Cookie Value:</B> ${cookie.JSESSIONID.value}
  <LI><B>Server:</B> ${pageContext.servletContext.serverInfo}
</UL>
</BODY></HTML>
```

Operators.jsp

```
<HTML>
<HEAD><TITLE>EL Operators</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  EL Operators
</TABLE>
<P>
<TABLE BORDER=1 ALIGN="CENTER">
```

```
  <TR><TH CLASS="COLORED" COLSPAN=2>Arithmetic Operators
      <TH CLASS="COLORED" COLSPAN=2>Relational Operators
  <TR><TH>Expression<TH>Result<TH>Expression<TH>Result
  <TR ALIGN="CENTER">
    <TD>\${3+2-1}<TD>${3+2-1}  <%-- Addition/Subtraction --%>
    <TD>\${1&lt;2}<TD>${1<2}      <%-- Numerical comparison --%>
  <TR ALIGN="CENTER">
    <TD>\${"1"+2}<TD>${"1"+2}     <%-- String conversion --%>
    <TD>\${"a"&lt;"b"}<TD>${"a"<"b"} <%-- Lexical comparison --%>
  <TR ALIGN="CENTER">
    <TD>\${1 + 2*3 + 3/4}<TD>${1 + 2*3 + 3/4}  <%-- Mult/Div --%>
    <TD>\${2/3 &gt;= 3/2}<TD>${2/3 >= 3/2}        <%-- >= --%>
  <TR ALIGN="CENTER">
    <TD>\${3%2}<TD>${3%2}                  <%-- Modulo --%>
    <TD>\${3/4 == 0.75}<TD>${3/4 == 0.75} <%-- Numeric = --%>
  <TR ALIGN="CENTER">
    <%-- div and mod are alternatives to / and % --%>
    <TD>\${(8 div 2) mod 3}<TD>${(8 div 2) mod 3}
    <%-- Compares with "equals" but returns false for null --%>
    <TD>\${null == "test"}<TD>${null == "test"}

  <TR><TH CLASS="COLORED" COLSPAN=2>Logical Operators
      <TH CLASS="COLORED" COLSPAN=2><CODE>empty</CODE> Operator
  <TR><TH>Expression<TH>Result<TH>Expression<TH>Result
  <TR ALIGN="CENTER">
    <TD>\${(1&lt;2) && (4&lt;3)}<TD>${(1<2) && (4<3)} <%--AND--%>
    <TD>\${empty ""}<TD>${empty ""} <%-- Empty string --%>
  <TR ALIGN="CENTER">
    <TD>\${(1&lt;2) || (4&lt;3)}<TD>${(1<2) || (4<3)} <%--OR--%>
    <TD>\${empty null}<TD>${empty null} <%-- null --%>
  <TR ALIGN="CENTER">
    <TD>\${!(1&lt;2)}<TD>${!(1<2)}  <%-- NOT -%>
    <%-- Handles null or empty string in request param --%>
    <TD>\${empty param.blah}<TD>${empty param.blah}
</TABLE>
   </BODY></HTML>
```

2.5 Expresii conditionale in EL

Expresiile se pot afisa in functie de evaluarea unei expresii la rulare:

${test ? expresie1 : expresie2}

Exemplu:

SalesBean.java

```
public class SalesBean {
  private double q1, q2, q3, q4;

  public SalesBean(double q1Sales,
                   double q2Sales,
                   double q3Sales,
                   double q4Sales) {
    q1 = q1Sales;
    q2 = q2Sales;
    q3 = q3Sales;
    q4 = q4Sales;
```

```
  }

  public double getQ1() { return(q1); }

  public double getQ2() { return(q2); }

  public double getQ3() { return(q3); }

  public double getQ4() { return(q4); }

  public double getTotal() { return(q1 + q2 + q3 + q4); }
}
```

ConditionalsServlet.java

```
@WebServlet("/conditionals")
public class ConditionalsServlet extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    SalesBean apples =
      new SalesBean(150.25, -75.25, 22.25, -33.57);
    SalesBean oranges =
      new SalesBean(-220.25, -49.57, 138.25, 12.25);
    request.setAttribute("apples", apples);
    request.setAttribute("oranges", oranges);
    RequestDispatcher dispatcher =
      request.getRequestDispatcher("/WEB-INF/results/conditionals.jsp");
    dispatcher.forward(request, response);
  }
}
```

Conditionals.jsp

```
<HTML>
<HEAD><TITLE>Conditional Evaluation</TITLE>
<LINK REL=STYLESHEET
      HREF="./css/JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
  Conditional Evaluation
</TABLE>
<P>
<TABLE BORDER=1 ALIGN="CENTER">
  <TR><TH>
      <TH CLASS="COLORED">Apples
      <TH CLASS="COLORED">Oranges
  <TR><TH CLASS="COLORED">First Quarter
      <TD ALIGN="RIGHT">${apples.q1}
      <TD ALIGN="RIGHT">${oranges.q1}
  <TR><TH CLASS="COLORED">Second Quarter
      <TD ALIGN="RIGHT">${apples.q2}
      <TD ALIGN="RIGHT">${oranges.q2}
  <TR><TH CLASS="COLORED">Third Quarter
```

```
      <TD ALIGN="RIGHT">${apples.q3}
      <TD ALIGN="RIGHT">${oranges.q3}
  <TR><TH CLASS="COLORED">Fourth Quarter
      <TD ALIGN="RIGHT">${apples.q4}
      <TD ALIGN="RIGHT">${oranges.q4}
  <TR><TH CLASS="COLORED">Total
      <TD ALIGN="RIGHT"
          BGCOLOR="${(apples.total < 0) ? "RED" : "WHITE" }">
      ${apples.total}
      <TD ALIGN="RIGHT"
          BGCOLOR="${(oranges.total < 0) ? "RED" : "WHITE" }">
      ${oranges.total}
</TABLE>
   </BODY></HTML>
```

3. Exercitii.

1) Creati o pagina *index.html* prin intermediul careia se apelati exemplele din laborator: Variabile scope, Proprietati bean, Colectii, Obiecte implicite, Operatori, Expresii conditionale .

2) Creati si testati un servlet care trimite spre afisare unui pagini jsp un vector cu 3 nume.

3) Creati un bean Persoana cu nume si prenume. Creati un servlet care trimite un vector cu 3 persoane pentru afisare intr-o pagina JSP. Alternativ folositi si ArrayList.

4) Creati un bean Angajat. Acesta are propietatile Persoana si Adresa. Persoana are nume si prenume, iar Adresa are strada, oras, cod postal. Creati un servlet care trimite spre afisare unei pagini JSP un vector de 5 angajati.