Aula 02 – 01/jun

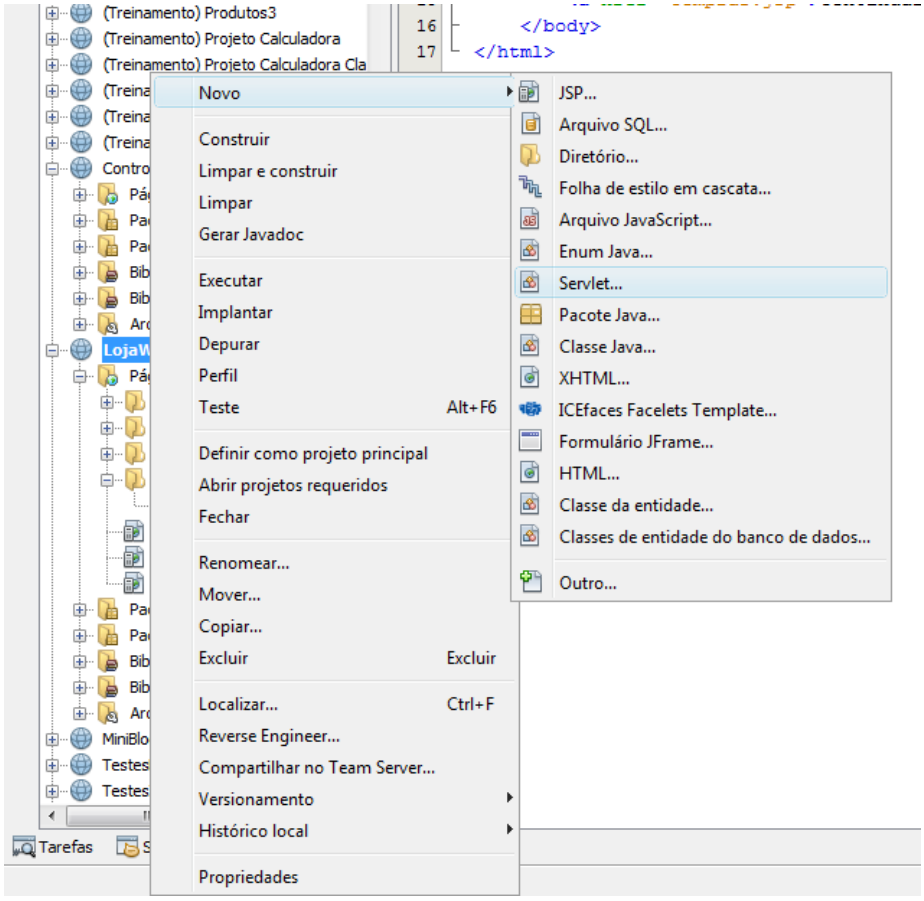
Servlets

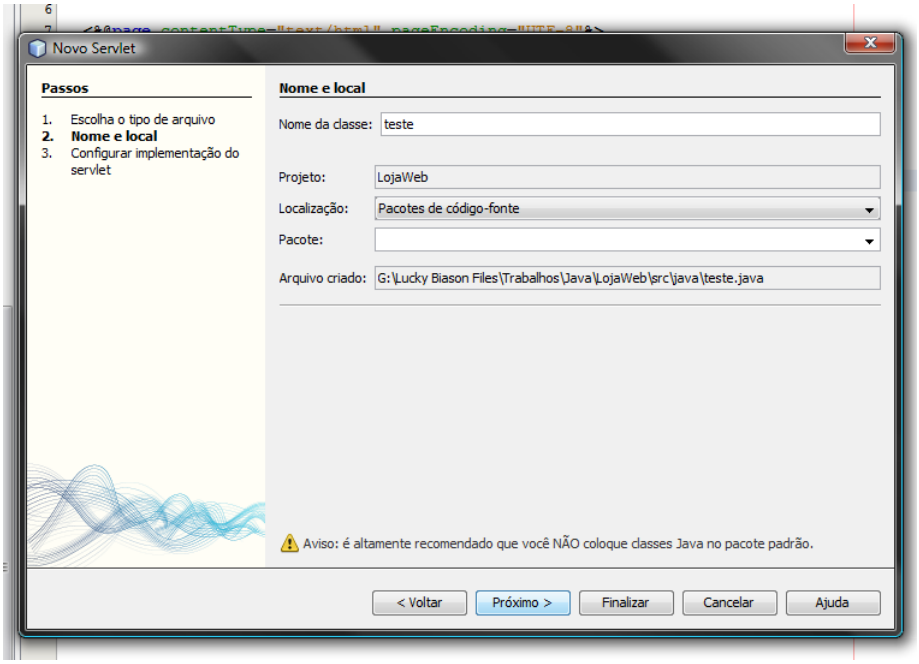
Formulários HTML podem receber entradas via caixa de texto, mas eles não conseguem processar ou usar essas informações para criar uma resposta dinâmica. Uma forma de processar informação de entrada é ter um **servlets** – trecho especial de código em Java que pode extrair informação de uma solicitação e enviar a resposta desejada de volta ao cliente.

Um servlets é simplesmente um tipo especial de classe Java que tem uma estrutura predefinida, atributos e métodos que são chamados em uma sequência predefinida e específica, toda vez que houver uma solicitação para o servlets.

Os dois mais importantes métodos de uma classe **servlets** são o **doPost()** e **doGet(),** que são chamados sempre que houver uma solicitação do tipo POST/GET para servlets, respectivamente.

Para criar um servlet, vá a Novo, Servlet. Defina um nome para ele, e na próxima tela (A MAIS IMPORTANTE!!), a configuração do servlets. Com isso ele será criado no XML principal para a aplicação WEB, e com isso ele poderá ser acessado facilmente.





Ele terá uma cara assim:

```
3
4  import java.io.IOException;
5  import java.io.PrintWriter;
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10
11 /**
12  *
13  * @author Lucas Biason
14  */
15 public class teste extends HttpServlet {
16
17
18     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
19     throws ServletException, IOException {
20         response.setContentType("text/html;charset=UTF-8");
21         PrintWriter out = response.getWriter();
22         try {
23             /* TODO output your page here
24             out.println("<html>");
25             out.println("<head>");
26             out.println("<title>Servlet teste</title>");
27             out.println("</head>");
28             out.println("<body>");
29             out.println("<h1>Servlet teste at " + request.getContextPath () + "</h1>");
30             out.println("</body>");
31             out.println("</html>");
32             */
33         } finally {
34             out.close();
35         }
36     }
37
38     HttpServlet methods. Click on the + sign on the left to edit the code.
73
74 }
75
```

Esse é toda a parte dele que você precisa se preocupar no começo. O resto cria métodos para receber as informações dos formulários, via POST ou GET, e envia para esse método.

Caso haja necessidade de um tratamento diferenciado no envio de informações, isto é, precise que o tratamento para caso recebe POST seja um e GET outro, dai a próxima parte

se torna muito importante.

Um exemplo simples seria fazer o servlets não aceitar o envio de informações via um dos dois, e retornar para a página anterior, ou outro exemplo seria, tratar com mais segurança o método GET, por ser mais vulnerável que o POST.

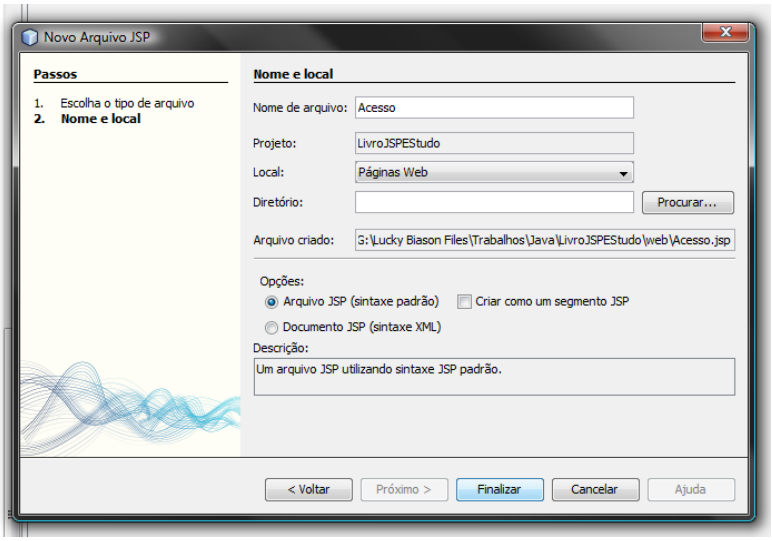
Os métodos doPost e do Get estão descritos aseguir:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}


@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
```

Dentro desses métodos você pode programar de forma diferenciada.

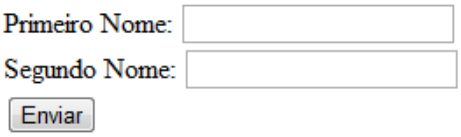
Exemplo de processamento de formulários usando um servlets

1º) Vamos criar uma nova JSP chamada Acesso.



2º) Vamos criar o conteúdo da página. Um pequeno formulário com o Primeiro e Ultimo Nome de uma Pessoa.

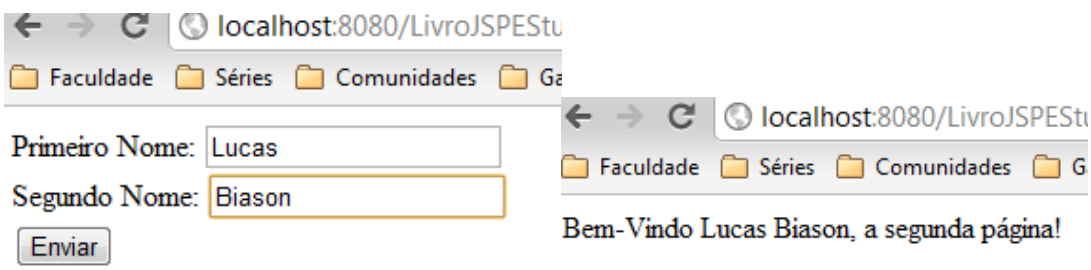
```jsp
<%--
    Document   : Acesso
    Created on : 03/05/2011, 18:49:03
    Author     : Lucas Biason
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<html>

    <body>
        <form name="principal" action="teste" method="post">
            Primeiro Nome: <input type="text" name="primeironome"><br/>
            Segundo Nome:  <input type="text" name="segundonome"><br/>
            <input type="submit" value="Enviar">
        </form>
    </body>
</html>
```



3º) Vamos criar um servlets para receber os parâmetros do formulário.

```java
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    String nome = request.getParameter("primeironome");
    String segundonome = request.getParameter("segundonome");

    PrintWriter out = response.getWriter();
    out.print("<html><body>"
            + "Bem-Vindo "
            + nome +" "+segundonome
            + ", a segunda página!"
            + "</body></html>");
    out.close();
}
```

Testando:



Sessão e Java Beans – Duas formas de modificar o escopo de um parâmetro.

Cada vez que você envia uma solicitação para o servidor, ele considera que você é um novo usuário. Ele não sabe que você é a mesma pessoa que acabou de solicitar algum outro URL alguns momentos atrás. É assim que o protocolo HTTP funciona e isto pode ser um problema em certas situações.

Por exemplo: digamos que você esteja conectado e comprando um item em um sitio de compras on-line. Quando você clicar em "Comprar" e seguir para examinar algum outro item, é fundamental que o servidor reconheça que você é a mesma pessoa que quer continuar comprando e receber uma conta global ao terminar. Um processo de compras normalmente se estende por umas poucas páginas o que significa que os valores informados pelo usuário na primeira página não estarão disponíveis na terceira página nem mais além. Da mesma forma, valores informados na segunda página não estarão disponíveis na quarta página nem mais além.

Para resolver esses problemas, é conveniente separar o código Java das páginas JSP, o que se pode fazer com o auxilio dos JavaBeans e da taglibs. Outro uso possível é o armazenamento de informações nos vários escopos da aplicação. Exemplo:

- application.setAttribute("nome","Júlio");

- session.setAttribute("nome","Júlio");

- request.setAttribute("nome","Júlio");

- page.setAttribute("nome","Júlio");

Escopo de objetos JSP:

Application: Define objetos que persistem durante toda a aplicação onde foram criados, até sua remoção do web container.

Session: Define objetos válidos durante a sessão de navegação do cliente.

Request: Define objetos acessíveis em todas as paginas envolvidas no processamento de uma requisição, até a finalização da resposta.

Page: Define objetos acessíveis apenas nas páginas onde foram criados.

Exemplo: Vamos criar uma jsp chamada inicio. A qual terá dois campos a ser preenchidos: nome e telefone. Ambos os campos, em vez de serem validados por JavaScript, serão por Java mesmo. Para isso é necessário criar uma página jsp vazia, que ira apenas receber os parâmetros e verificar se estão preenchidos. Caso não estejam, ela irá retornar a

pagina anterior. Caso estejam, ela irá para a próxima página.

Inicio.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<html>
    <head>
    </head>
    <body>
        <form name="principal" action="validar.jsp" method="post">
            Nome: <input type="text" name="nome"><br/>
            Telefone: <input type="text" name="telefone"><br/>
            <input type="submit" value="Enviar">
        </form>
    </body>
</html>
```

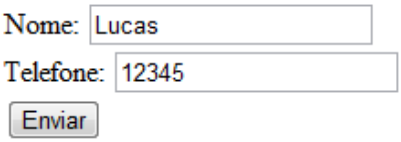
Validar.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%
    String nome = request.getParameter("nome");
    String telefone = request.getParameter("telefone");

    if(nome == null || nome.equals("")){
        response.sendRedirect("index.jsp");
    }else{
        response.sendRedirect("ProcessarAcesso.jsp");
    }

%>
```

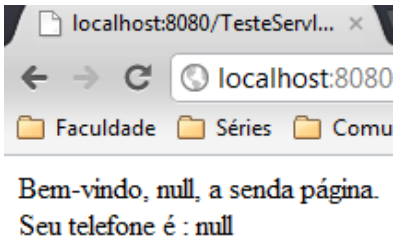
ProcessarAcesso.jsp:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<html>
    <body>
        Bem-vindo, <%= request.getParameter("nome") %>, a segunda página.<br/>
        Seu telefone é : <%= request.getParameter("telefone") %>
    </body>
</html>
```

O Iniciamos o projetos normalmente, e inserimos os valores no campo:



E o erro já aparece de cara:



Em vez de colocar o nome e o telefone digitados, ele coloca null. Isso por que os parâmetros nome e telefone, vivem apenas entre Acesso e Validar, pois estão no request. Como explicado acima, temos algumas opções:

a) Colocar campo "hidden" – ocultos em um formulário e enviar para a próxima página ou enviar os parâmetros via url. (deixa o sistema mais lento, ruim por que via url os parâmetros ficam visíveis. Existe muita redundância de informações.).
b) Usando servlets (visto mais a frente) (dificulta a programação e criação da pagina jsp).
c) Usar JavaBean (uma boa opção, mas o Javabeans é mais complicado que mexer em certos casos, nesse nem tanto)
d) Armazenar os parâmetros recebidos no objeto Session ou Application para que possam ser vistos por toda a aplicação. (vamos escolher esse por ser o mais fácil e também um método muito bom).

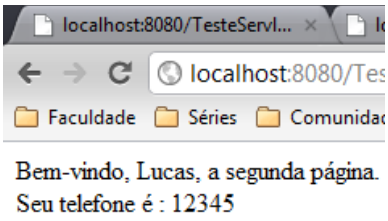
A página Validar ficará:

```
<%
    String nome = request.getParameter("nome");
    String telefone = request.getParameter("telefone");

    if(nome == null || nome.equals("") || telefone == null || telefone.equals("")){
        response.sendRedirect("index.jsp");
    }else{
        session.setAttribute("nome",nome);
        session.setAttribute("telefone",telefone);
        response.sendRedirect("ProcessarAcesso.jsp");
    }

%>
```

A página ProcessarAcesso ficara:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<html>
    <body>

        Bem-vindo, <%= session.getAttribute("nome") %>, a segunda página.<br/>
        Seu telefone é : <%= session.getAttribute("telefone") %>

    </body>
</html>
```

Com isso o problema foi resolvido:



JavaBeans – um estudo

O desenvolvimento de sistemas com JSP apresenta como problema principal a mistura de código e XHTML.

Em caso de alterações tanto programadores quanto web-designers devem ser envolvidos. A melhor solução é separar a lógica em classes designadas por Java Beans. Estas classes podem ser acedidas diretamente da página JSP através de uso de propriedades.

Como não há programação, a tarefa pode ser realizada pelo web designer diminuindo o impacto tanto da alteração de código quanto ao do layout.

Java Beans são classes Java que obedecem determinadas regras:

– Deve existir um construtor público e sem parâmetros

– Nenhum atributo pode ser público

– Os atributos são acedidos através de métodos públicos setXxx, getXxx

e isXxx.

Estas regras determinam um padrão que possibilita o uso de Beans como componentes em ferramentas de desenvolvimento. Estes componentes minimizam a necessidade de programação pois são utilizados através de suas propriedades.

Para utilizar Java Beans em uma aplicação comum deve-se criar um objeto e aceder

aos seus métodos; Em JSP, existem Marcas especiais para criação e recuperação de propriedades que não exigem conhecimento de programação:

<jsp:useBean>: Cria objetos Java ou seleciona um objeto que já existe para que seja possível utiliza-lo numa JSP

• Exemplo:

```
<jsp:useBean id = “aluno” class = “Aluno”/>
```

• Esta tag é semelhante a:

```
Aluno aluno = new Aluno();
```

Para ler uma propriedade de um Bean usa-se o atributo getProperty

• Exemplo:

```
<jsp:getProperty name = “aluno” property = “nome” />
```

Esta Marca retorna no local em que estiver o valor da propriedade recuperada.

<jsp:setProperty> : Para alterar uma propriedade.

• Exemplo:

```
<jsp:setProperty name = “aluno” property = “nome” value = “Maria”/>
```

Inicializar Beans: Caso seja necessário inicializar um Beans usa-se a sintaxe:

```
<jsp:useBean id = “aluno” class = “Aluno”>
        <%-- Inicialização do Bean --%>
</jsp:useBean>
```

O código é executado apenas se o Bean for criado

Propriedades Indexadas: Não existem Marcas específicas para o acesso a propriedades indexadas. Para aceder tais propriedades deve-se usar scriptlets e expressões.

• Exemplo:

```
<%for(int i=0; i<10; i++) { %>
<%=aluno.getPropriedade(i)%> <br>
<% } %>
```

Propriedades e Parâmetros: Os parâmetros (getParameter) podem ser inseridos diretamente em propriedades de Java Beans. Basta usar o nome do parâmetro no atributo

param:

```
<jsp:setProperty name="aluno" property="nome" param = "nome"/>
```

Para propriedades e parâmetros com o mesmo nome é possível fazer a associação total com o uso de "*"

Exemplo:

```
<jsp:setProperty name="aluno" property="*" />
```

A comparação dos nomes é sensível a maiúsculas e minúsculas!!!!

TagLibs, o JSTL

A possibilidade de criação de tags personalizadas levou a uma proliferação de TagLibs. Embora destinada a objetivos distintos, essa multiplicidade de bibliotecas trouxe duas situações inconvenientes: a duplicação de funcionalidades e a incompatibilidade entre TagLibs diferentes.

Para prover algum grau de padronização e evitar o dispêndio de esforça com tarefas repetitivas, propôs-se a JSTL (JSP Standard Tag Library), uma biblioteca padronizada de tags para atividades comuns em aplicações web, tais como execução de laços, controle de decisão, formatação de texto, acesso a bancos de dados, etc.

O objetivo da JSTL é permitir a criação de páginas JSP sem uso de scriptlet (isto é, sem a presença de código Java), empregando exclusivamente a estrutura de tags.

Vantagens: melhora a legibilidade do código, facilita a separação entre a lógica de negócios e a apresentação, provê reuso e possibilita maior grau de automação por parte das ferramentas de autoria.

Desvantagem: são menos flexíveis que os scriptlet e seu uso torna os servlets equivalentemente maiores e mais complexos.

A JSTL prove um conjunto de tags destinadas a realiza tarefas comuns, tais como: repetição, tomada de decisão, seleção, acesso a banco de dados, internacionalização e processamento de documentos XML. Ela se divide em quatro bibliotecas:

Core: tarefas comuns (saída, repetição, tomadas de decisão e seleção), sua URI é http://java.sun.com/jsp/jstl/core.

Database Access: acesso aos bancos de dados. A URI para seu uso é http://java.sun.com/jsp/jstl/sql.

Formating & I18N: contém tags destinadas a internacionalização e formatção de datas, moedas, valores e outros dados. Sua URI é: http://java.sun.com/jsp/jstl/fmt.

XML Processing: processamento de documentos XML. Sua URI é http://java.sun.com/jsp/jstl/x.

Core taglibs (apenas veremos parte dessa biblioteca. O restante estará disponível no site do Java:

Uso: <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

Saída básica: Permite a saída de mensagens e expressões e tem a seguinte sintaxe:

```
<c:out
        value="conteúdo da mensagem"
        default="conteúdo alternativo"
         escapeXML="<true|false>"
/>
```

*escapeXML indica o uso dos caracteres especiais.

Variáveis: definir ou ajustar o valor de uma variável no escopo indicado.

```
<c:set
        var="nomeVar"
        target="nomeTarget"
        property="nomeProperty"
        value="exprEL"
        scope="<request|page|session|application>"
/>
```

*var - nome da variável para armazenar valor.
*target - noma da variável para modificar valor (requer uso de property).
* value - expressão que dá valor a variável.
*scope - indica o escopo da variável var (default=page).

Decisões: avaliação condicional.

```
<c:if
        Test="condição"
        Var="nome"
        scope="<request|page|session|application>">
        <%-- código executado quando expressão EL resulta true --%>
</c:if>
```

Repetições: repete a execução de um bloco de código conforme estabelecido por sua variável.

- Repetição comum:

```
<c:forEach
        Var=”nome”
        Begin=”valor de inicio”
        End=”valor de final”
        Step=”passo”>
<%-- corpo a ser repetido conforme controle --%>
</c:forEach>
```

- repetição sobre conjuntos:

```
<c:forEach
        Var=”nome”
        Items=”<array|Collection|Map|Iterator|Enumeration>”
        varStatus=”nome”>
<%-- corpo a ser repetido conforme controle --%>
</c:forEach>
```

Projetos feitos em aula:

-TesteServlets

- TesteEscopo

- TesteAcaoJavaBean

- Produtos

- Produtos 2 (Estudo para casa)

Exercícios Complementares de fixação:

1- Refaça os exercícios feitos em sala.
2- Estude o Projeto Produtos 2.
3- Refaça o exercício 2 complementar da aula 01. Use servlets para receber os parâmetros, e envie para uma página de resposta.
4- Refaça o exercício 2 complementar da aula 01. Use sessões e valida os campos utilizando JSP. (conforme foi visto do exemplo sobre sessões).
5- Refaça o exercício 2 complementar da aula 01. Use JavaBeans para passar o objeto para uma terceira pagina e mostre os valores do objeto na página.
6- Refaça o exercício 2 complementar da aula 01. Misture o uso de Sessões, JavaBeans e Taglibs.
7- Troque as estruturas de controle (if, for, while...) nos exercício anteriores por taglibs.
8- Refaça os exercícios 3 e 4 na aula01 nas mesmas condições propostas nos

exercícios de 3 a 7 dessa aula.

Exercícios de Pesquisa:

1. Defina Servlets, Javabeans e tagLIbs.
2. Pesquisa sobre as 4 grandes bibliotecas do JSTL.
3. Pesquise sobre Custom Tags.
4. Pesquise mais sobre Sessões e sobre Cookies.