Módulo 5

Desenvolvimento de Aplicações Móveis

Lição 7

Comunicação Corporativa

Versão 1.0 - Set/2007

Autor
xxx

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Anderson Moreira Paiva
Andre Neves de Amorim
Angelo de Oliveira
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Fábio Bombonato
Fabrício Ribeiro Brigagão
Francisco das Chagas
Frederico Dubiel
Herivelto Gabriel dos Santos
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Kefreen Ryenz Batista Lacerda
Kleberth Bezerra G. dos Santos
Leandro Silva de Morais
Leonardo Ribas Segala
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Seire Pareja
Sergio Terzella
Vanessa dos Santos Almeida
Robson Alves Macêdo

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Nesta lição, aprenderemos a escrever *Servlets*, utilizar páginas com *JSP* e *JSTL*, acessar um banco de dados através de JDBC e a criar e ler documentos *XML*.

Ao final desta lição, o estudante será capaz de:

- Escrever *Servlets* simples
- Escrever *Java Server Pages* (JSP) simples utilizando JSTL
- Escrever código JSTL utilizando *Expression Language* (EL)
- Acessar banco de dados utilizando a tecnologia JDBC
- Gerar XML
- Ler um arquivo XML no cliente móvel

2. Servlets

Servlet é uma classe Java que é executada no servidor WEB para implementar transações do tipo requisição-resposta (*request-response*). Através dos *servlet*, a tecnologia Java fornece uma alternativa melhor e mais portável do que *scripts CGI* (*Common Gateway Interface*). Comparado com *scripts CGI* (escritos em Perl, *scripts sheel*, ou outras linguagens de script), *servlet* fornece uma tecnologia escalável, independente de plataforma, para entregar conteúdo dinâmico.

Um *servlet* tem dois métodos para processar uma requisição, os métodos *doGet()* e *doPost()*. Esses métodos são chamados quando um cliente WEB (*browser*) envia um comando "*GET*" ou "*POST*". Esses métodos têm parâmetros idênticos. Esses parâmetros são normalmente entregues para um método comum. Portanto, tanto a requisição *GET* quanto a requisição *POST* são manipuladas da mesma maneira.

```
protected void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
  // requisição de processamento ...
}

protected void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
  // requisição de processamento ...
}
```

A seguir teremos um *servlet* completo:

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SampleServlet extends HttpServlet {

  protected void processRequest(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet SampleServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet SampleServlet at "
      + request.getContextPath () + "</h1>");
    out.println("</body>");
    out.println("</html>");
    out.close();
  }
  protected void doGet(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    processRequest(request, response);
  }
  protected void doPost(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    processRequest(request, response);
  }
  /** Retorna uma breve descrição do servlet.
   */
```

```
    public String getServletInfo() {
      return "Short description";
    }
  }
```

Para criar um novo Projeto WEB (*Web Project*) no NetBeans, selecione a partir do menu principal a opção *File -> New Project*...

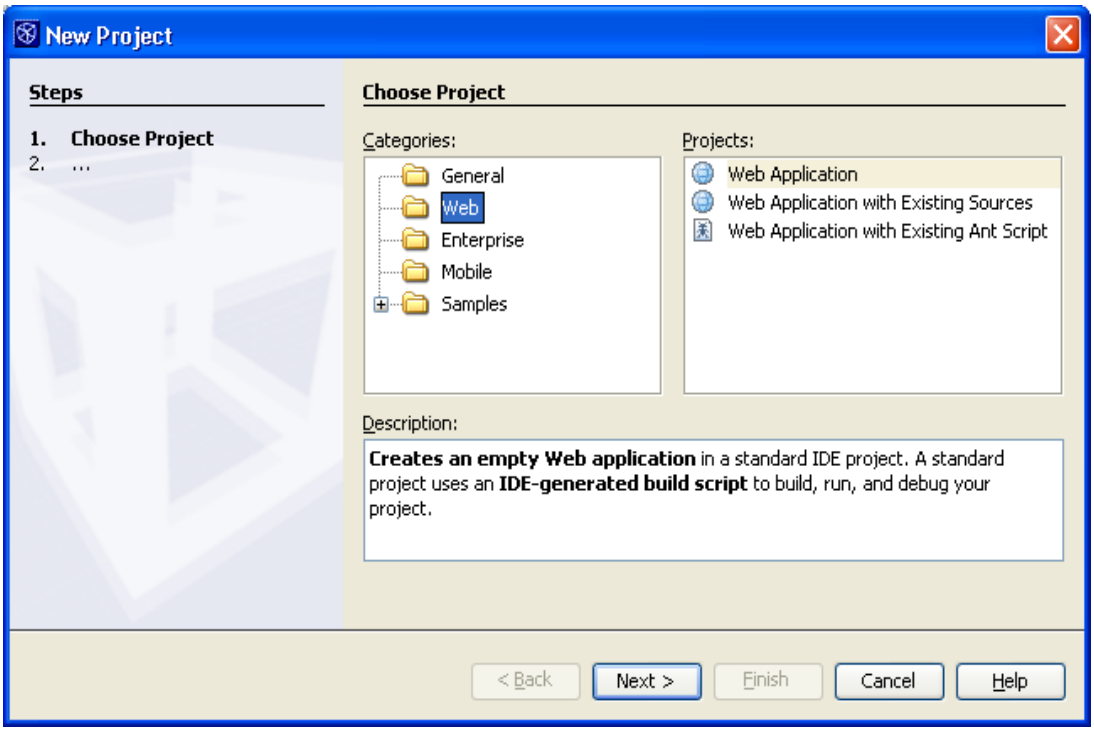

Figura 1: Janela New Project do NetBeans

Na janela acima selecione em *Categories* a opção *Web* e em *Projects* selecione a opção *Web Application*.

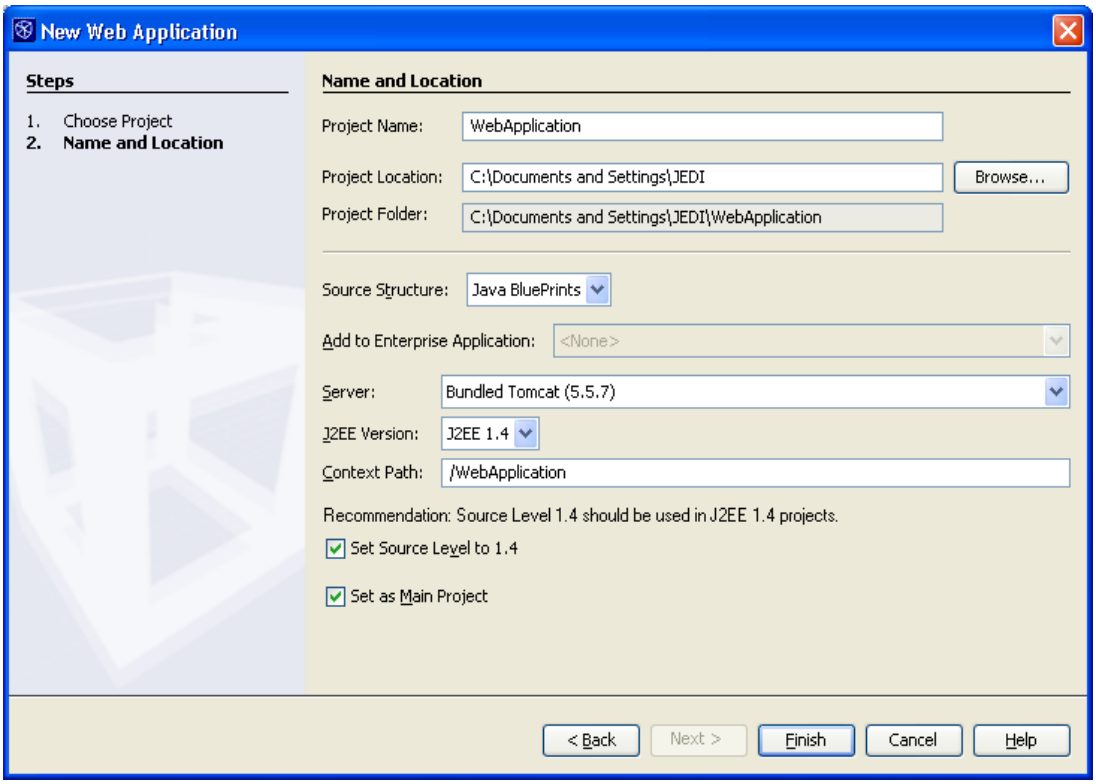

Figura 2: Janela New Web Application do NetBeans

Na janela acima informe o nome para o projeto e pressione o botão *Finish*. Após o projeto criado poderemos criar um novo arquivo tipo *servlet*. Para o mesmo. Para criar este arquivo selecione, a partir do menu principal, a opção em *File -> New File...*

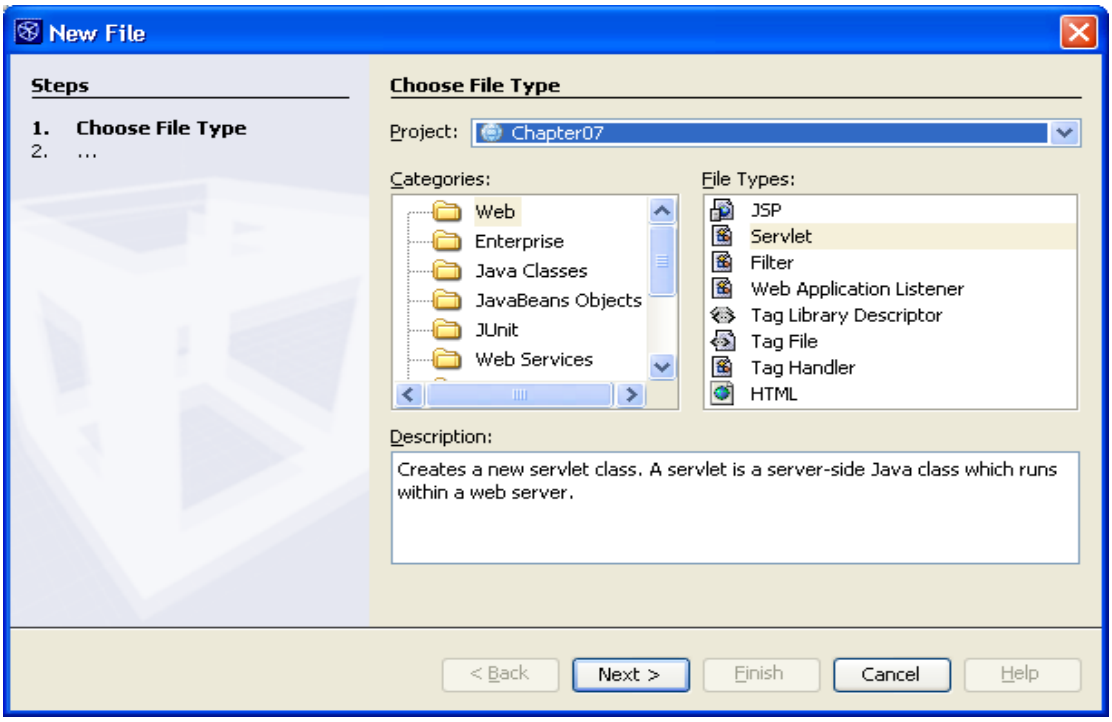

Figura 3: Janela New File do NetBeans

Na janela acima selecione em *Categories* a opção *Web* e em *File Types* selecione a opção *Servlet*.

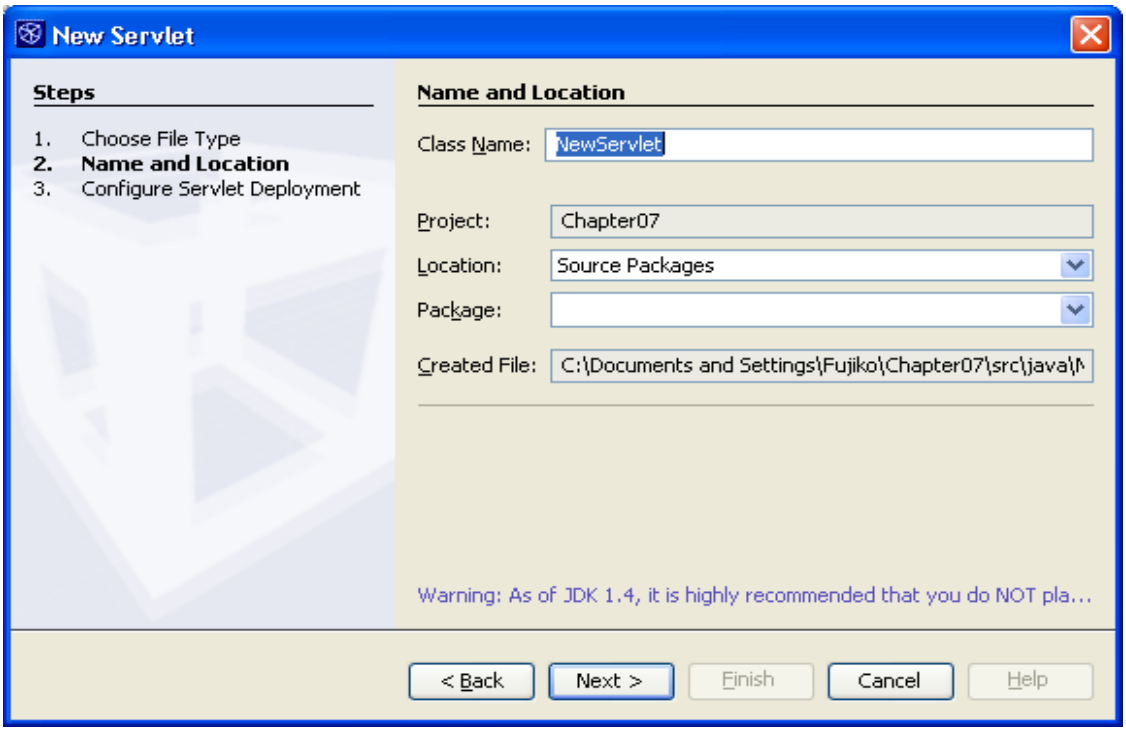

Figura 4: Janela New Servlet do NetBeans

Na janela acima informe o nome para o *servlet* e pressione o botão *Finish*.

3. JSP e JSTL

O objetivo primário das *Java Server Pages* (JSP) e *Standard Tag Library* (JSTL) é ajudar aos desenvolvedores simplificar a escrita das de páginas WEB. *JSTL* faz a ponte entre programadores e autores (não programadores) fornecendo uma linguagem de expressão simples para a construção de páginas JSP.

Além do suporte às ações da linguagem de expressão e de fluxo de controle, *JSTL* fornece também funcionalidade para acessar recursos baseados em URL, internacionalização e formatação de números e datas, acesso de base de dados e processamento de XML.

JTSL é composta de diversas bibliotecas de *tags*, agrupadas com base na área funcional.

Área	*Prefixo*	*URI*	*Exemplo de código*
core	c	http://java.sun.com/jstl/core	<c:out value="${var}"/>
I18N formatting	fmt	http://java.sun.com/jstl/fmt	<fmt:formatNumber
XML processing	x	http://java.sun.com/jstl/xml	<x:forEach ...
Database (SQL)	sql	http://java.sun.com/jstl/sql	<sql:query var=...
Functions	fn	http://java.sun.com/jstl/functions	<fn:

Nesta seção, abordaremos o uso da biblioteca *core*.

3.1. Configuração

Para usar as tags JSTL, a diretiva da taglib deve ser incluída na página JSP, uma para cada área funcional que será usada na página.

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

```
<%@page pageEncoding="UTF-8"%>

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

Figura 5: Código JSP com a tag core

Deve-se incluir também o **jstl.jar** no diretório de bibliotecas do projeto:

1. Pressionar o botão direito do mouse em cima do diretório de bibliotecas do projeto e selecionar "Add Library" no menu pop-up:

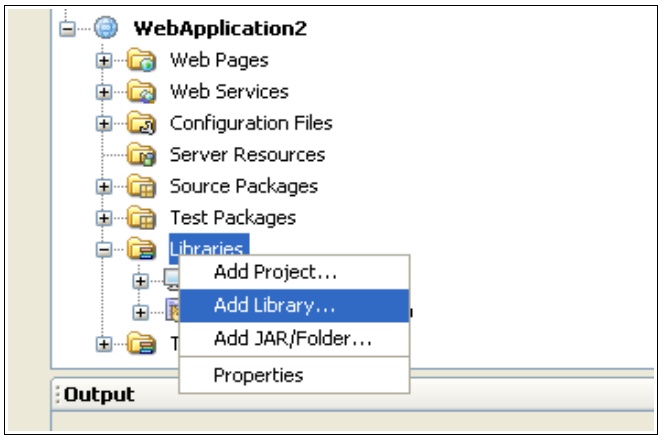

Figura 6: Adicionando uma biblioteca ao projeto

2. Selecionar "JSTL 1.1" e selecionar a opção "Add Library". **"JSTL 1.1 – standard.jar"** e **"STL 1.1 – jstl.jar"** serão adicionados à biblioteca do projeto.

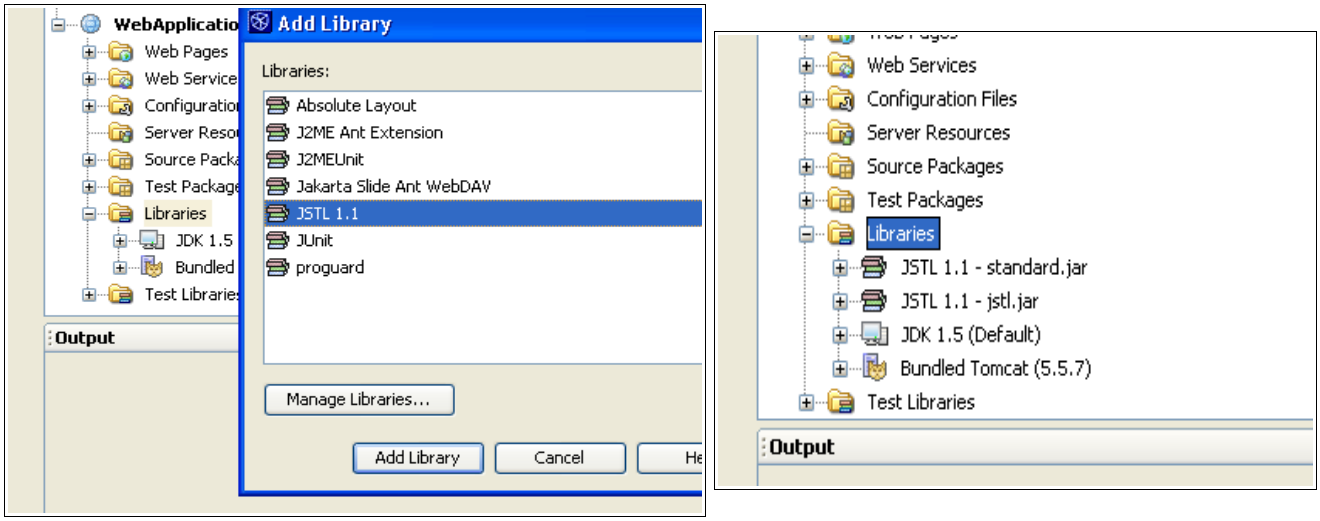

Figura 7: Adicionando uma biblioteca ao projeto

3.2. Hello, world!

Uma página JSP, ao contrário de um *servlet*, é uma página HTML com *tags* que permitem a programação Java. A seguir temos uma página JSP:

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

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<c:set var="mensagem" value="hello, world!"/>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title><c:out value="${mensagem}"/></title>
    </head>
    <body>
    <h1><c:out value="${mensagem}"/></h1>
    </body>
</html>
```

Seguindo as mesmas instruções para gerar um *servlet*, entretanto selecione JSP e indique o nome como **hello.jsp**. Para ver a saída desta página, pressionar o botão direito do mouse em cima do nome do arquivo e selecionar a opção "Run File" é mostrado o seguinte resultado no seu navegador web:

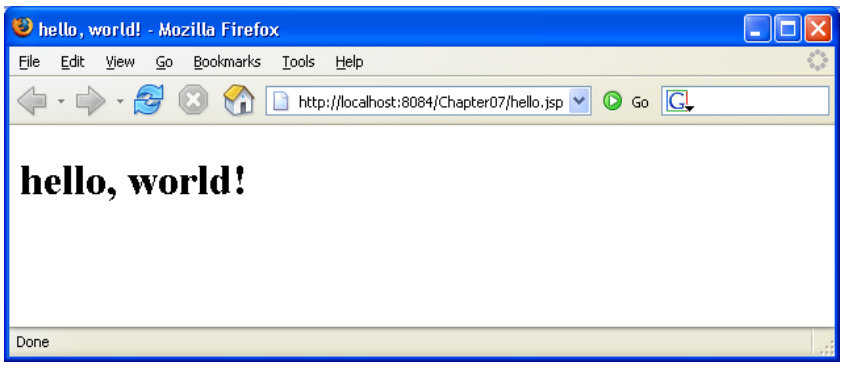

Figura 8: Resultado da execução da página JSP

3.3. Transferindo controle do Servlet para JSP

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SampleServlet2 extends HttpServlet {

    protected void processRequest(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String[] days = {"Sunday", "Monday", "Tuesday", "Wednesday"};
        request.setAttribute("days", days);
        request.setAttribute("message", "hello, world!");
        // Transfere o controle para a página JSP
        RequestDispatcher dispatcher = request.
              getRequestDispatcher("/hello.jsp");
        if (dispatcher != null)
            dispatcher.forward(request, response);
    }
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
    protected void doPost(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        processRequest(request, response);
    }
    public String getServletInfo() {
        return "Short description";
    }
}
```

Figura 9: Resultado da execução do Servlet

3.4. Linguagem de Expressão

3.4.1. Acessando Atributos, Propriedades e Coleções

Para acessar um atributo, a sintaxe é: `${var}`

Exemplo: saída do valor do atributo username

```
<c:out value="${username}"/>
```

JSTL unifica o acesso às propriedades do JavaBean e valores de coleção. A expressão varX.varY é equivalente para varX[varY] em JSTL. A expressão varX.varY (ou varX[varY]) será avaliada, dependendo do tipo de varX:

1. Se varX é um JavaBean, varY será convertido em cadeia de caractere. Se varY é uma propriedade de leitura de varX, ela pode retornar o resultado da chamada ao método de acesso: varX.getVarY()
2. Se varX é uma coleção do tipo List: varY é forçada para int. Converter para: varX.get(varY)
3. Se varX é uma coleção do tipo Vector: varY é forçada para int. Converter para: Array.get(varX, varY)
4. Se varX é uma coleção do tipo Map: Converter para: varX.get(varY)

Os atributos podem ter escopo de página, de requisição e da aplicação. O linguagem de expressão procuraria pelo identificador nestes escopos. Se o identificador não for achado, é retornado nulo.

3.4.2. Objetos Implícitos

JSTL inicializa automaticamente diversos atributos de mapeamento com valores de origens diferentes. Estes atributos estão disponíveis para as páginas JSP sem qualquer inicialização do usuário. Por exemplo, o atributo "param" contém mapeamentos de nomes de parâmetros e valores de requisição. Estes nomes e valores (param) vêm de formulários HTML, através da de submissão de métodos HTTP GET ou POST.

Objeto Implícito	***Conteúdo***
pageScope	Contém um mapeamento de nomes de atributos de escopo de página para seus valores
requestScope	Contém um mapeamento de nomes de atributos de escopo de requisição para seus valores
sessionScope	Contém um mapeamento de nomes de atributos de escopo de sessão para seus valores
applicationScope	Contém um mapeamento de nomes de atributos de escopo de aplicação para seus valores
param	Contém um mapeamento de nomes de parâmetros para seus valores de parâmetros (cadeia de caracteres). Equivalente a *ServletRequest.getParameter*(String)
header	Contém um mapeamento de nomes de cabeçalho para seus valores (cadeia de caracteres). Equivalente a s*ervletRequst.getHeader*(String)
cookie	Contém contendo um mapeamento de nomes de "cookies" para seus valores. Equivalente a *HttpServletRequest.getCookie*(String)

3.4.3. Operadores

A linguagem da expressão (EL) da JSTL suporta operadores relacionais, aritméticos e lógicos. Os operadores relacionais suportados são:

- `==` ou **`eq`**
- `!=` ou **`ne`**

- < ou **lt**
- > ou **gt**
- <= ou **le**
- >= ou **ge**

Os operadores lógicos suportados são:

- && ou **and**
- || ou **or**
- ! ou **not**

E os operadores aritméticos suportados são:

- + (adição)
- - (subtração)
- * (multiplicação)
- / (divisão)
- % ou **mod** (resto da divisão ou módulo)

O operador adicional **empty** é muito útil para testar valores nulos ou vazios.

```
<c:if test="${empty param.username}">No username</c:if>
```

3.4.4. Exceções a Valores Padrões (Básicos)

Para simplificar páginas de JSP, os erros simples não gerarão exceções. Indicar um atributo com um valor nulo indicará simplesmente "(zero)" em vez de gerar um **NullPointerException**.

```
Username: <input
          type="text"
          value="<c:out value="${param.username}"/>"
          name="username" />
```

Qualquer atributo não definido que for utilizado em expressões terá seu valor padrão como 0 (zero). Esta expressão retornaria o valor 1, se o parâmetro "start" não fosse inicializado:

```
<c:out value="${param.start + 1}"/>
```

3.5. Biblioteca Core

3.5.1. Tag <c:out>

O tag <c:out> avalia uma expressão e retorna o seu resultado.

Sintaxe :

```
<c:out
  value="value"
  [escapeXml="{true|false}"]
  [default="defaultValue"]
/>
```

Nome	*Dinâmico*	*Requisito*	*Tipo*	*Descrição*
value	sim	sim	Objeto	A expressão a ser avaliada
escapeXml	sim	não	boolean	Se verdadeiro, os caracteres **<**, **>**, **&**, **'** e **"** são convertidos em seus códigos da entidade do caráter (por exemplo: > conversos ao >). O valor padrão é **true**.
default	sim	não	Objeto	O valor padrão se o valor do resultado for nulo

Exemplos:

```
Rows: <c:out value="${param.numRows}" defaultValue="20" />
Description:
<pre>
  <c:out value="${bookmark.description}" escapeXml="false" />
</pre>
```

3.5.2. Tag <c:set>

Ajusta o valor de um atributo em um determinado escopo.

Sintaxe:

```
<c:set
  value="value"
  var="varName"
  [scope="{page|request|session|application}"]
/>
```

Nome	*Dinâmico*	*Requisito*	*Tipo*	*Descrição*
value	sim	sim	Objeto	A expressão a ser avaliada
var	não	sim	String	O nome do atributo exportada que conterá o valor da expressão. O tipo do atributo segue o tipo do resultado da expressão
scope	não	não	String	O escopo do atributo

Exemplo:

```
<c:set var="fullName" value="${lastName, firstName}" />
```

3.5.3. Tag <c:remove>

Esta *tag* remove um atributo de um determinado escopo.

Sintaxe:

```
<c:remove
  var="varName"
  [scope="{page|request|session|application}"]
/>
```

Nome	*Dinâmico*	*Requisito*	*Tipo*	*Descrição*
var	não	sim	String	O nome do atributo a ser eliminado
scope	não	não	String	O escopo do atributo

3.5.4. Tag <c:if>

Realiza uma avaliação condicional. O conteúdo do corpo será processado se o teste de avaliação da condição informada for verdadeiro.

Sintaxe:

```
<c:if test="testCondition"
  [var="varName"] [scope="{page|request|session|application}"]>
  body content
</c:if>
```

ou

```
<c:if test="testCondition"
  var="varName" [scope="{page|request|session|application}"]
/>
```

Nome	Dinâmico	Requerido	Tipo	Descrição
test	sim	sim	*boolean*	A condição testada que determina se o conteúdo do corpo deverá ser processado
var	não	não/sim	String	Nome do atributo exportado que irá conter o valor da condição testada. O tipo do atributo de escopo é *Boolean*
scope	não	não	*String*	Escopo do atributo

Exemplo:

```
<c:if test="${empty param.username}">No username</c:if>
```

3.5.5. Tag <c:choose><c:when><c:otherwise>

A tag <*c:choose*> é uma substituta para a instrução *if-else-if* do Java, permitindo a execução condicional mutuamente exclusiva. O conteúdo do corpo composto pela tag <*c:otherwise*> será avaliado se nenhuma instrução das tags <*c:when*> for considerada verdadeira. O bloco <*c:otherwise*> deverá ser o último da instrução e também ser precedido por no mínimo uma tag do tipo <*c:when*>.

Sintaxe:

```
<c:choose>
  <c:when test="condition1">
    instruções para esta condição
  </c:when>
  <c:when test="condition2">
    instruções para esta condição
  </c:when>...
  <c:otherwise>
    instruções caso nenhuma condição tenha sido considerada verdadeira
  </c:otherwise>
</c:choose>
```

Nome	Dinâmico	Requerido	Tipo	Descrição
test	sim	sim	*boolean*	A condição de teste que determina se o conteúdo do corpo deverá ser processado

Exemplo:

```
<c:choose>
  <c:when test="${gender eq 'M'}">Male</c:when>
  <c:when test="${gender eq 'F'}">Female</c:when>
  <c:otherwise>Unknown</c:otherwise>
</c:choose>
```

3.5.6. Tag <c:forEach>

A tag <*c:forEach*> realiza iteração sobre o conteúdo de uma coleção. A coleção pode ser qualquer uma das subclasses de *java.util.Collection* e *java.util.Map*. *Arrays* de objetos e tipos primitivos também são suportados. Uma String com valores separados por vírgula ("*True,False*") também pode ser utilizada para o processo. Enquanto houver itens na coleção o conteúdo do corpo será processado.

A tag também poderá ser utilizada para iterações com número fixo de repetições.

O atributo *varStatus* é do tipo *javax.servlet.jsp.jstl.core.LoopTagStatus* e tem a propriedade *index* (índice da iteração atual, iniciado em zero) e *count* (a contagem da iteração atual, iniciada em 1 (um), por exemplo, se o índice inicial é 20, o índice final é 80 e o intervalo é 10, o *count* poderá ser 1,2,3,4,5,6,7). As propriedades lógicas *first* e *last* indicam se a iteração atual é a primeira ou

a última, respectivamente. Existem também as propriedades *begin*, *end* e *step* que guardam os valores dos argumentos inicial, final e do passo realizado pelo laço.

Sintaxe:

```
<c:forEach
  [var="varName"]
  items="collection"
  [varStatus="varStatusName"]
  [begin="begin"] [end="end"] [step="step"]
>
  instruções a serem executadas
</c:forEach>
```

ou

```
<c:forEach [var="varName"]
  [varStatus="varStatusName"]
  begin="begin" end="end" [step="step"]
>
  instruções a serem executadas
</c:forEach>
```

Nome	*Dinâmico*	*Requerido*	*Tipo*	*Descrição*
var	não	não	variável	O nome de um atributo exportado para cada elemento de uma coleção
items	sim	sim/não	Collection, Map, Array, String	Coleção de itens para realizar interações
varStatus	não	não	String	O nome do atributo exportado para a situação da operação. O tipo do objeto exportado é *javax.servlet.jsp.jstl.core.LoopTagStatus*
begin	sim	não/sim	int	Índice inicial (base zero) na coleção, ou um índice inicial fixo (se a coleção não for especificada)
end	sim	não/sim	int	Índice final (base zero) na coleção, ou um índice final fixo (se a coleção não for especificada)
step	sim	não	int	O laço será processado por um intervalo definido de itens da iteração

Exemplo:

```
<select name="gender">
<c:forEach var="gender" items="${genderList}">
  <option value="<c:out value="${gender.code}"/>">
    <c:out value="${gender.name}"/>
  </option>
</c:forEach>
</table>
```

3.5.7. Tag <c:forTokens>

Esta tag realiza iteração sobre os segmentos de texto que devem ser separados por um delimitador em um objeto do tipo String.

Sintaxe:

```
<c:forTokens
  items="stringOfTokens"
  delims="delimiters"
  [var="varName"]
  [varStatus="varStatusName"]
  [begin="begin"] [end="end"] [step="step"]
>
```

```
  instruções a serem executadas
</c:forTokens >
```

Nome	***Dinâmico***	***Requerido***	***Tipo***	***Descrição***
var	não	não	String	O nome do atributo exportado para toda instrução iteração e símbolo
items	sim	sim	String	Literal de símbolos
delims	sim	sim	String	Delimitadores, caracteres que separam os símbolos no literal de itens
varStatus	não	não	String	O nome do atributo exportado para o status da operação. O objeto exportado é do tipo *javax.servlet.jsp.jstl.core.LoopTagStatus*
begin	sim	não	int	Índice do começo (base-zero) da iteração. Se não for especificado, a iteração vai começar com o primeiro símbolo.
end	sim	não	int	Final do índice da iteração
step	sim	não	int	O laço será processado por um intervalo definido de itens da iteração

Exemplo:

```
<select name="gender">
<c:forEach var="gender" items="Male,Female" delims=",">
  <option value="<c:out value="${gender}"/>">
    <c:out value="${gender}"/>
  </option>
</c:forEach>
</table>
```

4. JDBC

Nessa seção, discutiremos como persistir dados ou objetos. Para essa funcionalidade precisamos de um banco de dados (relacional). A biblioteca JDBC permite executar consultas e alterações em um banco de dados. Antes de podermos usar o JDBC, precisamos ter certeza de que três pré-requisitos estão sendo satisfeitos:

- JDBC *library* (biblioteca) – incluído no JDK
- O servidor de banco de dados – usaremos o MySQL (*www.mysql.com*)
- *Driver JDBC* – vem com o DBMS, instale o jar mysql-connector-java-3.x.x-bin.jar para o JDBC 3.0

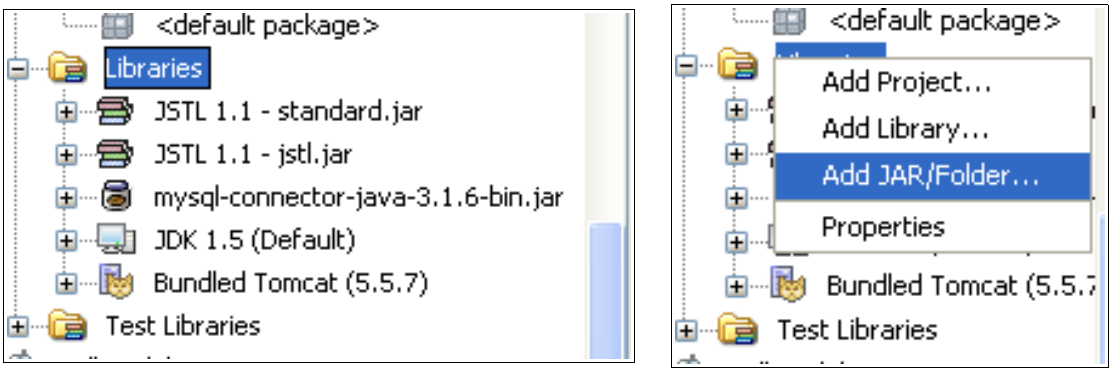

Figura 10: Adicionando as bibliotecas ao projeto

A tabela usada nesses exemplos pode ser recriada usando os comandos SQL CREATE AND INSERT:

```
CREATE TABLE `task` (
  `id` bigint(20) unsigned NOT NULL auto_increment,
  `task` varchar(128) NOT NULL default '',
  `duration` int(11) NOT NULL default '0',
  `assignedTo` varchar(64) NOT NULL default '',
  `status` char(1) NOT NULL default '',
  PRIMARY KEY  (`id`)
);

INSERT INTO `task`
(`id`, `task`, `duration`, `assignedTo`, `status`)
VALUES
(1,'connect to database',2,'alex','0'),
(2,'list table rows',4,'alex','0'),
(3,'update row',8,'you','0');
```

4.1. Carregando o Driver

Para utilizar o *driver* JDBC com um banco de dados em particular temos que carregá-lo utilizando Class.forName(). O nome do *driver* é dependente do *driver* do banco de dados que carregaremos. Em nosso caso, utilizaremos o mysql jdbc *driver*:

```
String driver = "com.mysql.jdbc.Driver";
Class.forName(driver);
```

4.2. Estabelecendo a Conexão

Para estabelecer uma conexão com o banco de dados precisamos da URL para o banco de dados. Precisaremos também ter acesso ao banco de dados. Um nome de usuário e senha válidos para o acesso ao banco de dados serão requeridos.

```
String url = "jdbc:mysql://localhost:3306/jedi";
String username = "root";
```

```
String password = "password";
conn = DriverManager.getConnection(url, username, password);
```

4.3. Executando consultas SQL

O método *executeQuery()* retorna um objeto do tipo *ResultSet*. Para percorrer por todas as linhas do resultado da consulta utilizaremos o método *next*(). Existem alguns métodos que retornam as colunas da linha corrente, cada uma para um tipo de dados específico. Nesse exemplo, recuperamos atributos dos tipos *String* e *int* utilizando *getString*() e *getInt*():

```
Statement statement = conn.createStatement();
String query = "SELECT task,duration,duration FROM task";
ResultSet rs = statement.executeQuery(query);
out.println("<table>");
out.println("<tr>");
out.println("<th>Task</th>");
out.println("<th>Duration</th>");
out.println("<th>Assigned to</th>");
out.println("</tr>");
while (rs.next()) {
  String task = rs.getString("task");
  int duration = rs.getInt("duration");
  String assignedTo = rs.getString("assignedTo");
  out.println("<tr>");
  out.println("<td>" + task + "</td>");
  out.println("<td>" + duration + "</td>");
  out.println("<td>" + duration + "</td>");
  out.println("</tr>");
}
out.println("</table>");
```

Task	Duration	Assigned to
connect to database	2	alex
list table rows	4	alex
update row	8	you

Figura 11: Resultado da consulta

4.4. Alterando tabelas

Para modificar registros nas tabelas com os comandos *INSERT, UPDATE e DELETE* (inclusão, alteração e exclusão, respectivamente), o método *executeUpdate()* é utlizado.

```
String task = (String) request.getParameter("task");
String duration = (String) request.getParameter("duration");
String assignedTo = (String) request.getParameter("assignedTo");
String status = (String) request.getParameter("status");
Long id = new Long(idStr);
String updateQuery;
ResultSet rs = dao.query("SELECT id from task WHERE id ='" + id + "'");
if (rs.next()){
    // altera a entrada da tarefa
    updateQuery = "UPDATE task SET"
    + " task='" + (task != null? task:"") + "'"
    + ",duration='" + (duration != null ? duration:"") + "'"
    + ",assignedTo='" + (assignedTo != null ? assignedTo:"") + "'"
    + ",status='" + (status != null ? status:"") + "'"
    + " WHERE id=" + id;
} else {
    // nova entrada da tarefa
    updateQuery = "INSERT INTO task (task, duration, assignedTo, status) "
    + "VALUES ("
```

```
        + "'" + task + "',"
        + "'" + duration + "',"
        + "'" + assignedTo + "',"
        + "'" + status + "'"
        + ")";
    }
    statement.executeUpdate(updateQuery);
```

5. XML

XML, a linguagem de marcação extensível (*eXtensible Markup Language*), é uma linguagem de marcação baseada em texto. Com XML, pode-se apresentar dados em um documento estruturado de texto.

Assim como o HTML, as *tags* XML são definidas usando os símbolos maior e menor: <>. Entretanto, diferente do HTML, XML é mais fácil de se analisar. Um documento XML é estruturado com entidades formando uma estrutura de árvore.

Pode-se utilizar qualquer nome de *tag* apropriado que seja desejado, desde que todas as aplicações que utilizam o documento XML utilizem os mesmos nomes de *tag*. *Tags* podem conter atributos. No exemplo abaixo, a primeira "task" (tarefa) tem um atributo "id" (identificador) igual a "1" enquanto a segunda "task" tem um atributo "id" igual a "2".

```
<tasks>
  <task id="1">
    <name>connect to database</name>
    <duration>2</duration>
    <assignedTo>alex</assignedTo>
    <status>0</status>
  </task>
  <task id="2">
    <name>list table rows</name>
    <duration>4</duration>
    <assignedTo>alex</assignedTo>
    <status>4</status>
  </task>
</tasks>
```

5.1. Analisando o XML

Até a data da escrita deste texto, não havia nenhuma biblioteca padrão definida pelo *JCP* para análise de XML em CLDC. Entretanto, existem muitas bibliotecas XML que trabalham com a CLDC. Uma delas é a NanoXML (http://nanoxml.sourceforge.net/). A versão original da NanoXML não trabalha com a CLDC. O usuário deve baixar a versão modificada que trabalha com CLDC em http://www.ericgiguere.com/nanoxml e incluí-la no seu projeto móvel sob o nome de pacote "nanoxml".

```
import java.io.*;
import java.util.*;
import nanoxml.*;

...
   public String[] parseXml(String xml) {
     kXMLElement root = new kXMLElement();
     try {
       root.parseString(xml);
       Vector taskList = root.getChildren();
       Vector items = new Vector();
       for (int i=0; i<taskList.size(); i++){
         kXMLElement task = (kXMLElement) taskList.elementAt(i);
         String tagName = task.getTagName();
         if (tagName != null && tagName.equalsIgnoreCase("task")){
           Vector taskProperties = task.getChildren();
           String[] fieldNames = {"name", "duration", "assignedTo", "status"};
           String[] fields = new String[fieldNames.length];
           String id = task.getProperty("id", "0");
           for (int j=0; j<taskProperties.size(); j++) {
             kXMLElement prop = (kXMLElement) taskProperties.elementAt(j);
             String propTagName = prop.getTagName();
             if (propTagName != null) {
               for (int p=0; p<fieldNames.length; p++) {
                 if (propTagName.equalsIgnoreCase(fieldNames[p])) {
```

```
                    fields[p] = prop.getContents();
                  }
                }
              }
            }
            items.addElement(id + ": " + fields[0]);
          }
        }
        String[] itemArr = new String[items.size()];
        items.copyInto(itemArr);
        return itemArr;
      } catch( kXMLParseException ke ){
        return(null);
      }
    }
```

6. Exercícios

6.1. Cabeçalhos das tabelas com linhas de cores alternadas

Escrever um código JSTL que irá interagir com o mapa implícito "header" e mostrar o header chave/nome e o valor em uma tabela HTML. Linhas de número ímpar tem um fundo na cor lightyellow (<tr bgcolor="lightyellow">...). Note que o índice varStatus começa em zero. Note que o (header) Map tem as propriedades "key" e "value".

Saída do exemplo:

accept-encoding	gzip,deflate
connection	keep-alive
accept-language	en-us,en;q=0.5
host	localhost:8084
accept-charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
user-agent	Mozilla/5.0 (Linux; U; Linux v0.99; en-US) Gecko/20050511 Firefox/1.2.3
accept	text/xml,application/xml,application/xhtml +xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
keep-alive	300

6.2. Servlets e JSP

Criar um servlet e uma aplicação JSTL que deve mostrar no formato XML em um array de objetos. Os atributos do objeto são: nome e endereço IP. A classe Java deve parecer com esta:

```
public class Host {
  private String name;
  private String ip;
  public Host(String name, String ip) {
    this.name = name;
    this.ip = ip;
  }
  public String getName(){ return(name); }
  public String getIp(){ return(ip); }
  public void setName(String name){ this.name = name; }
  public void setIp(String ip){ this.ip = ip; }
}
```

Deve se passar um array estático dos Hosts no Servlet para o JSP usando o método *request.setAttribute*().

```
Host[] hosts = {
    new Host("localhost", "127.0.0.1"), new Host("java.sun.com", "1.2.3.4")};
```

A saída em XML se deve parecer como esta:

```
<hosts>
  <host name="localhost">
     <ip>127.0.0.1</ip>
  </host>
  <host name="java.sun.com">
    <ip>1.2.3.4</ip>
  </host>
</hosts>
```

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.