Módulo 6

Programação WEB

Lição 2

Classes Servlets

Versão 1.0 - Nov/2007

Autor
Daniel Villafuerte

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
Angelo de Oliveira
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Emanoel Tadeu da Silva Freitas
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Sergio Terzella
Thiago Magela Rodrigues Dias
Vanessa dos Santos Almeida
Wagner Eliezer Roncoletta

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

Uma *Servlet* é uma classe Java usada para estender a capacidade dos servidores que hospedam aplicações acessadas via modelo de programação Requisição/Resposta. É uma classe Java que implementa a interface Servlet e aceita requisições que vêm de outras classes Java, clientes Web ou outros Servlets, gerando, então, respostas.

As *Servlets* também são conhecidas como *HTTP Servlet*. Isto porque os Servlets são comumente usados com o HTTP atualmente, não há um protocolo cliente-servidor específico.

Para iniciar o uso das *Servlets* será necessário ter conhecimentos sobre programação Java, conceitos sobre cliente-servidor, HTML básico e HTTP (*HyperText Transfer Protocol*). Para criar uma *Servlet* será necessário importar para a nossa classe Java as classes de extensão padrões que estão dentro dos pacotes **javax.Servlet** e **javax.Servlet.http**.

O pacote ***javax.servlet*** contém a estrutura básica de *Servlet*, enquanto que o pacote **javax.Servlet.http** é utilizado como uma extensão da tecnologia para as Servlets que realizam requisições HTTP.

Ao final desta lição, o estudante será capaz de:

- Obter uma visão geral da arquitetura *Servlet*
- Conhecer o ciclo de vida de uma *Servlet*
- Manipular requisições e respostas
- Configurar, empacotar e distribuir uma aplicação WEB
- Conhecer os parâmetros de aplicações WEB

2. Conceitos Iniciais

2.1. Visão Geral da Arquitetura Servlet

Antes da tecnologia *Servlet*, o meio mais comum de adicionar funcionalidades a um servidor WEB era através do uso de **CGI** (*Common Gateway Interface* ou Interface de Entrada Comum). CGI fornece uma interface do servidor para uma classe externa permitindo que esta invoque o servidor a tratar as requisições do cliente. Porém, o CGI foi desenhado de forma que cada chamada para um recurso CGI fosse criado um novo processo no servidor; informação significante para a classe é passada para este processo utilizando entradas padrões e variáveis de ambiente. Uma vez completada a requisição, o processo é terminado, devolvendo o recurso ao sistema. O problema que incorre neste tipo de abordagem é que isto impõe pesadas requisições aos recursos do sistema limitando o número de usuários que a aplicação pode atender ao mesmo tempo.

A Tecnologia *Servlet* foi projetada para contornar este problema inerente ao CGI e prover aos desenvolvedores uma solução robusta em Java para criar aplicações WEB. Ao invés de criar um processo peso-pesado no servidor a cada requisição do cliente, com *Servlets* há somente um processo para todas as requisições: o processo solicitado pelo contêiner da *Servlet* para executar. Quando uma nova requisição chega, o contêiner cria somente uma pequena thread para executar a *Servlet*.

Servlets também são carregados em memória somente uma vez, ou seja, o contêiner carrega-os em memória na inicialização do servidor ou na primeira vez que o Servlet for requisitado para atender a um cliente, diferente do CGI, que para cada requisição do cliente carrega e descarrega os dados da classe em memória. Uma vez carregado em memória, está pronto para tratar as requisições dos clientes.

2.2. Primeira vista sobre Servlet

O código a seguir mostra a estrutura de uma *Servlet* básica que trata as requisições GET, assim como exibe o tradicional exemplo 'Hello World'.

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

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    //"request" é utilizado para ler os dados do formulário HTML dos cabeçalhos
    // HTTP (um exemplo são os dados digitados e submetidos)
    // e outros dados que podem ser obtidos a partir da requisição do cliente.

    // "response" é para especificar a linha e o cabeçalho de resposta do HTTP
    // (exemplo: especificar o tipo do conteúdo, definir cookies).
    // Também contém métodos que permitem ao Servlet gerar respostas para
    //  o cliente.

    PrintWriter out = response.getWriter();
    out.println("<HTML> <TITLE>Hello Page</TITLE><BODY><br>");
    out.println("<h1>Hello World!</h1>");
    out.println("</BODY></HTML>");

    //"out" para enviar o conteúdo para o browser.
  }
}
```

A primeira parte do código trata da importação das classes em java.io (para PrintWriter, etc.), javax.Servlet e javax.Servlet.http. Os pacotes **javax.servlet** e **javax.servlet.http** fornecem

interfaces e classes para escrever as *Servlets* (contêm as classes HttpServlet, HttpServletRequest e HttpServletResponse).

Por extensão da *HttpServlet*, herda os métodos que são automaticamente chamados pelo servidor dependendo de certas condições que serão explicadas mais à frente. Por polimorfismo por *overriding* destes métodos podemos fazer com que nossa *Servlet* execute a funcionalidade que quisermos.

Neste caso, o método herdado do HttpServlet que sobrepomos é o método *doGet*. Simplificando, ele é o método que é invocado pelo contêiner sempre que uma requisição GET é feita de uma *Servlet* em particular. Relembrando do módulo anterior, navegação de site, recuperação de documento, visualização de páginas são exemplos de requisições GET. Logo, sempre que um usuário quiser ver a resposta da nossa *Servlet* deve invocar uma requisição GET.

Ao visualizarmos o código veremos que o método *doGet* utiliza dois parâmetros: um objeto *HttpServletRequest* e um objeto *HttpServletResponse*. Não obstante, estes objetos vêm de encontro com as necessidades do desenvolvedor. São criados e mantidos por um *contêiner* e são simplesmente passados no momento que este *contêiner* chama o método *doGet*. A respeito disso, o método *doGet* e outros métodos, que serão discutidos depois, são similares ao método *public statis void main(Strings[] args)* que utilizamos em classes Java baseadas em linhas de comando. Não será criado um array de String para passarmos com o método; pois já foi realizado pelo ambiente de execução.

Os objetos *HttpServletRequest* e *HttpServletResponse* atendem aos desenvolvedores através das seguintes funcionalidades:

- O objeto HttpServletRequest fornece acesso a todas as informações a respeito das requisições do cliente incluindo todos os parâmetros que são colocados nos cabeçalhos de requisições HTTP, métodos de requisições HTTP, entre outros.
- O objeto HttpServletResponse contém todos métodos necessários utilizados pelos desenvolvedores para produzir respostas que serão enviadas de volta ao cliente. Isto inclui métodos para definir o cabeçalho de resposta HTTP, declarar respostas do tipo MIME, bem como métodos de recuperação de instâncias de classes Java I/O as quais podemos utilizar diretamente para produzir a saída.

Voltando ao código, vemos que, com a exceção dos comentários, existem muitas linhas que usamos para executar a funcionalidade de exibir "Hello World!" ao usuário. Uma dessas linhas é:

```
PrintWriter out = response.getWriter();
```

e múltiplas chamadas para o método *out.println().* Por enquanto, pensemos no objeto *out* como quem leva nosso texto de saída para o *browser* do cliente. Com isto em mente, é fácil ver como múltiplas chamadas ao método *out.println()* produz o seguinte conteúdo:

Figura 1: Saída do HelloServlet

2.3. Testando o exemplo Servlet

Até agora vimos a exibição de uma saída em uma *Servlet* exemplo. Para iniciarmos a abstração da implantação e configuração de *Servlets* faremos fazer uso de uma ferramenta automatizada disponibilizada por uma IDE. A IDE que utilizaremos utilizar em nosso exemplo é o Sun Studio Enterprise 8, que é livre para os membros do *Sun Developer Network*. Possui suporte completo para a tecnologia *Servlets* e especificações *JSP*, bem como outras características adicionais.

Daqui para frente assumiremos que o *Enterprise 8* foi instalado com sucesso em seu computador.

Antes de tudo, para o nosso Servlet exemplo, precisaremos criar um novo projeto de aplicação WEB. Para fazer isto, na barra de menu, selecione **New -> Project**. Na tela que aparece em seguida, selecione a categoria **Web**. Na direita, escolha **New Web Application**.

A tela seguinte irá solicitar detalhes de nosso projeto. Para nosso primeiro teste com *Servlets* usaremos como nome "FirstServletProject" (*Project Name*), e utilizaremos os valores padrões para os outros campos.

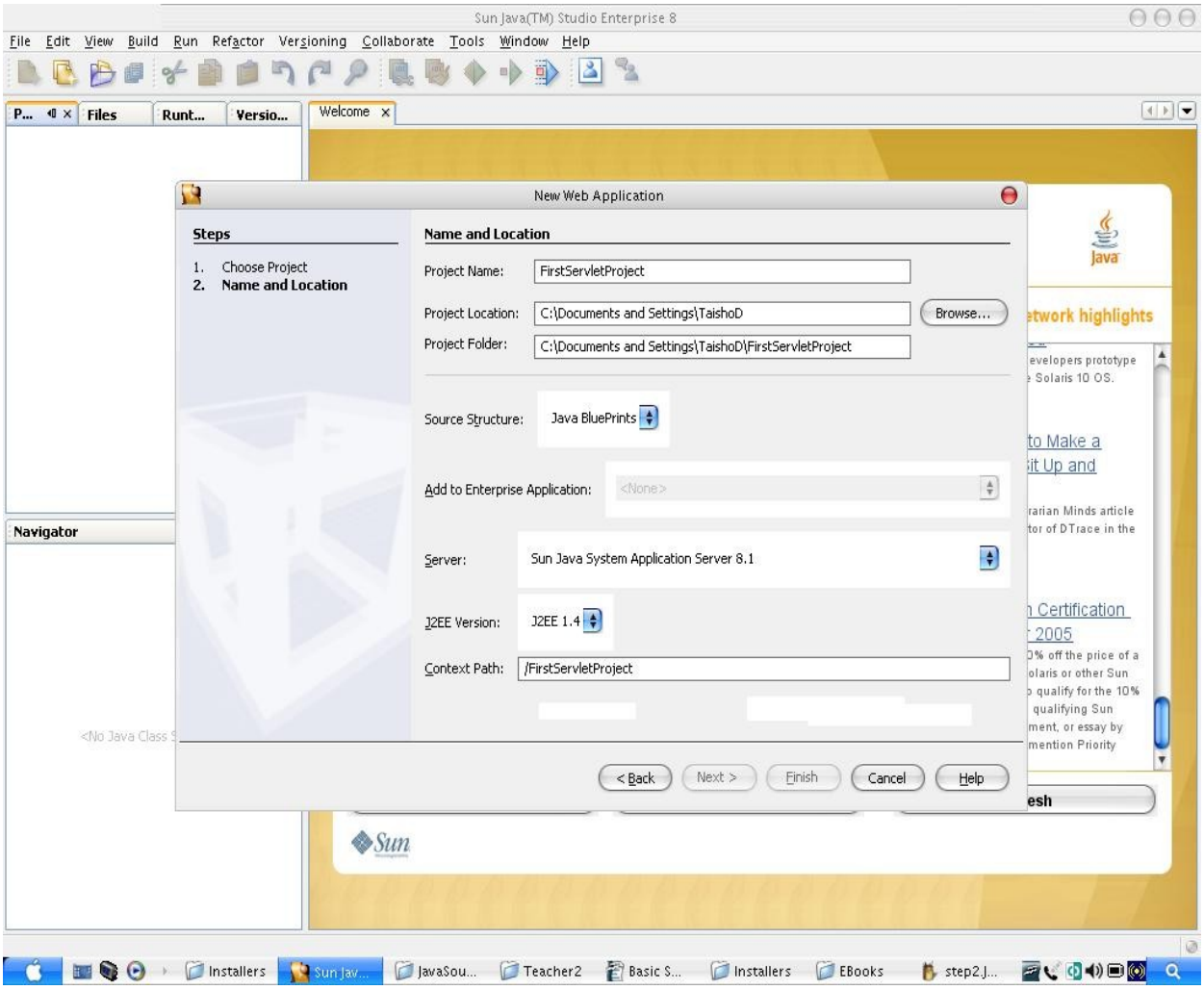

Figura 2: Criando um Novo Projeto de Aplicação Web

Após criar um novo projeto WEB, a tela será semelhante a figura a seguir:

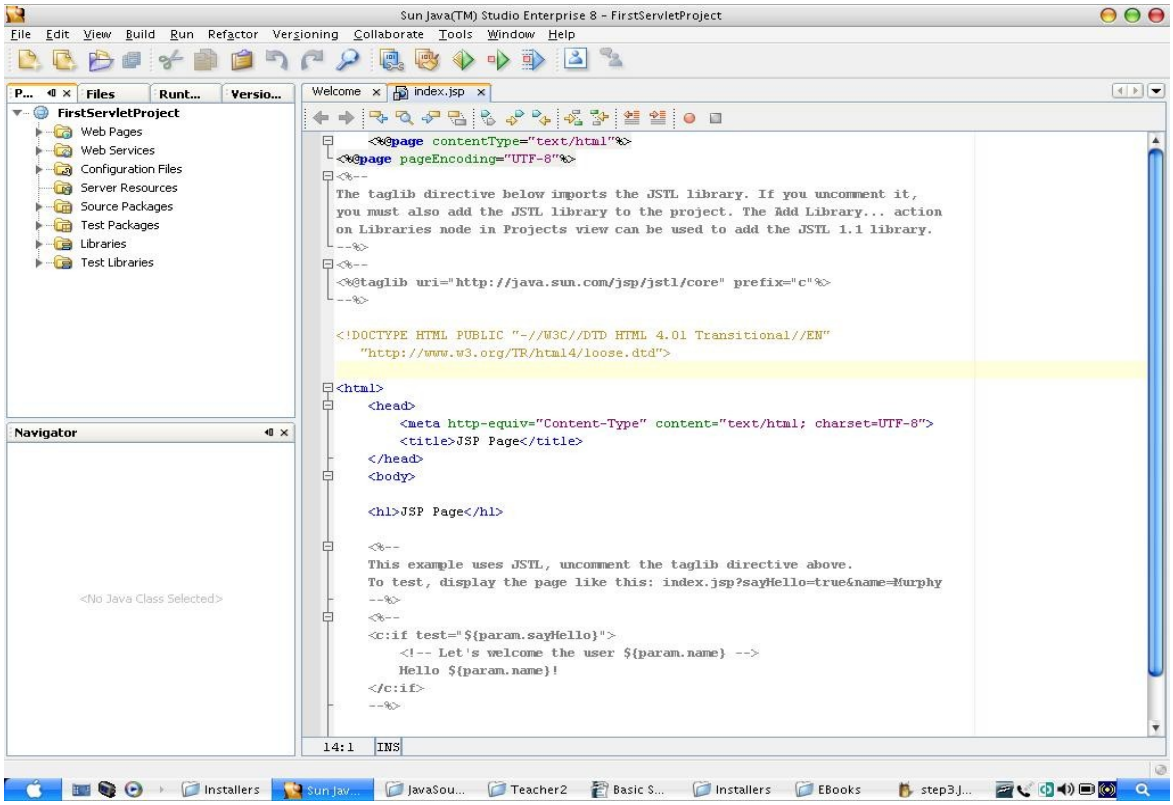

Figura 3: Novo Projeto de Aplicação Web

Para adicionar a nossa *Servlet* à aplicação, pressione o botão direito do mouse na opção **Source Packages**, selecione **New -> Servlet**. Se a *Servlet* não aparecer no menu, então selecione **New -> File/Folder**. Na tela seguinte, selecione a categoria **Web** e depois **Servlet**.

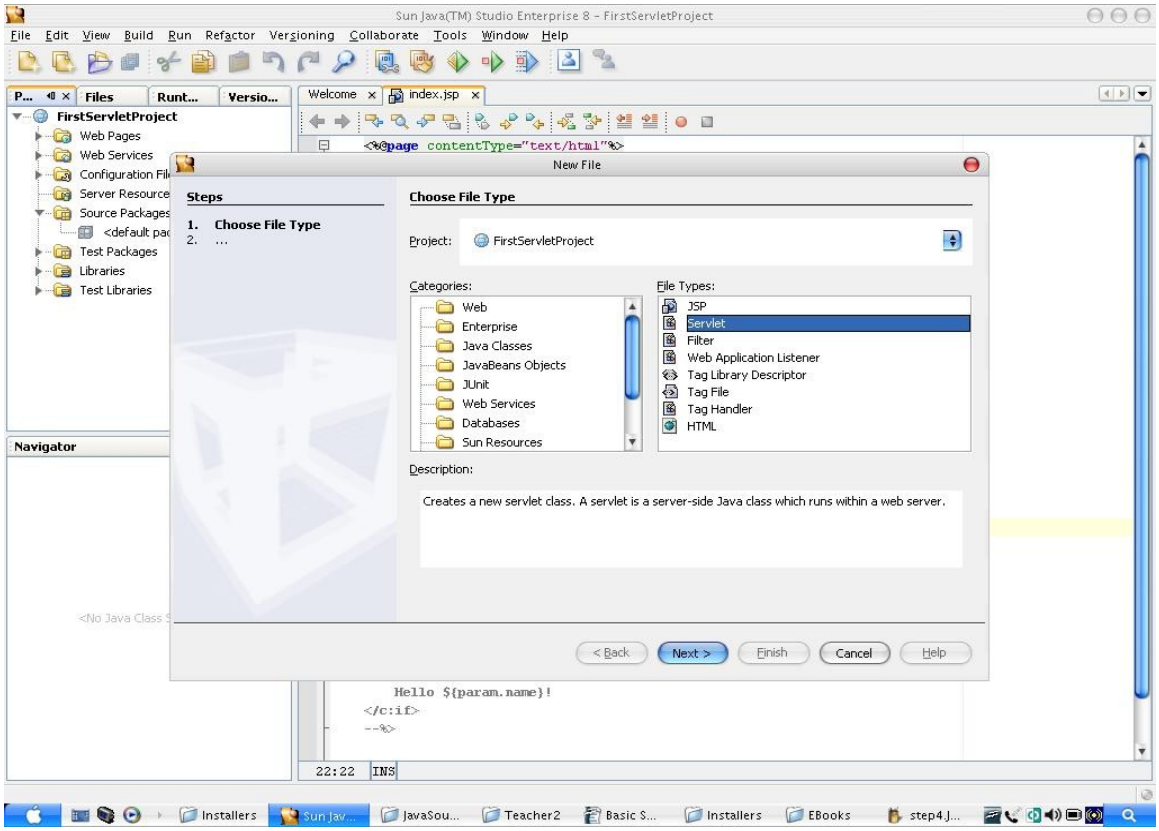

Figura 4: Adicionando Servlet para a aplicação

A IDE irá mostrar uma série de telas que questionarão sobre os detalhes da *Servlet* a ser criada. Na primeira tela, em **Class Name** digite ***FirstServlet***. Em **Package** digite **jedi.Servlet**.

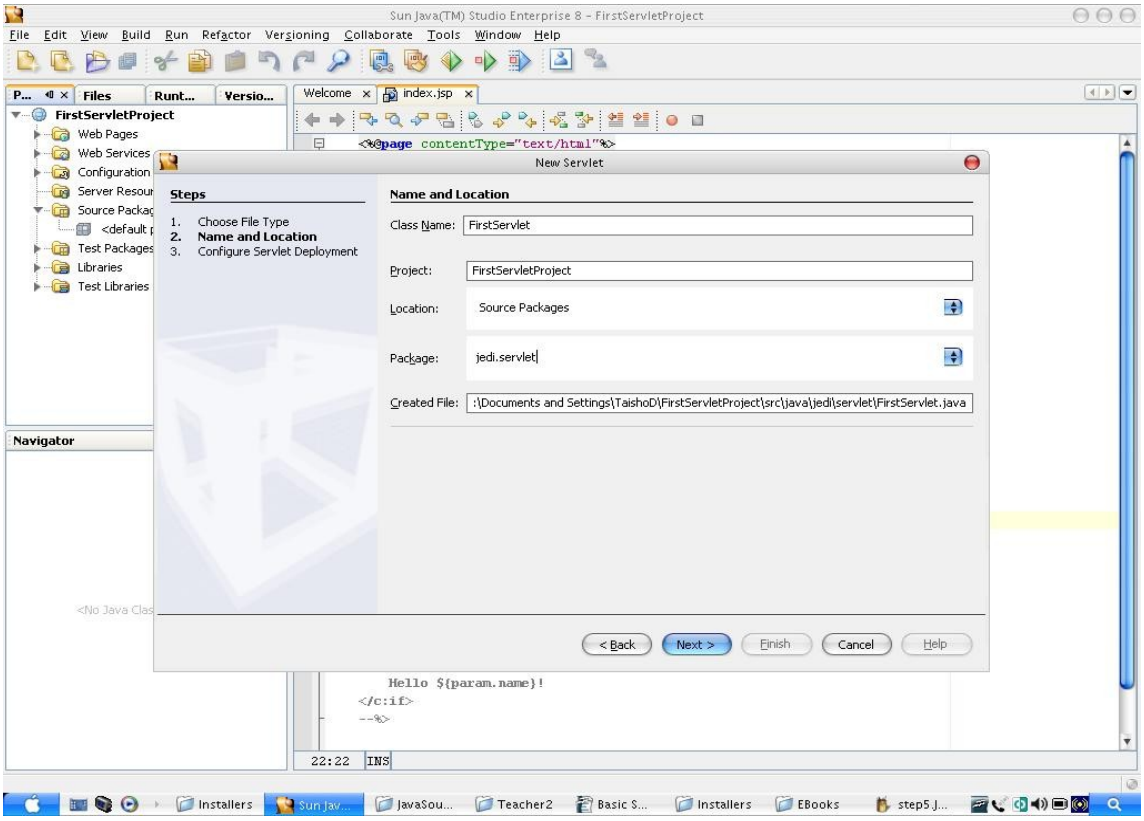

Figura 5: Modificando o nome da classe e o nome do pacote

Na tela que segue, mantenha os valores padrões e então pressione o botão **Finish** e isto irá resultar conforme a figura a seguir.

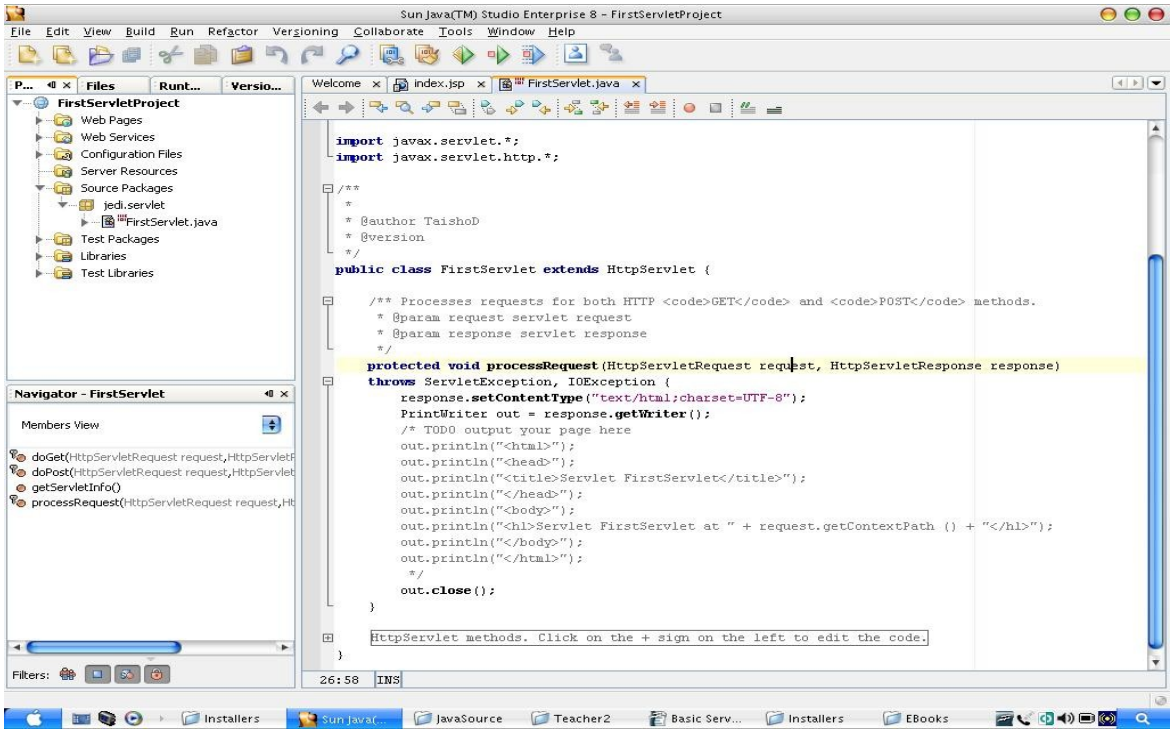

Figura 6: Servlet criada

Podemos perceber que a IDE criou e parcialmente implementou o método ***processRequest***. Se clicarmos na caixa com um sinal de soma na parte esquerda inferior veremos que o *processRequest* é simplesmente um método que é chamado pelos métodos ***doGet*** e ***doPost***. Isto significa que o conteúdo do método processRequest forma a base da funcionalidade de nosso Servlet.

Primeiro, removeremos todo o conteúdo do método *processRequest*. Então, copiaremos a nossa *Servlet* exemplo para dentro do método *doGet*.

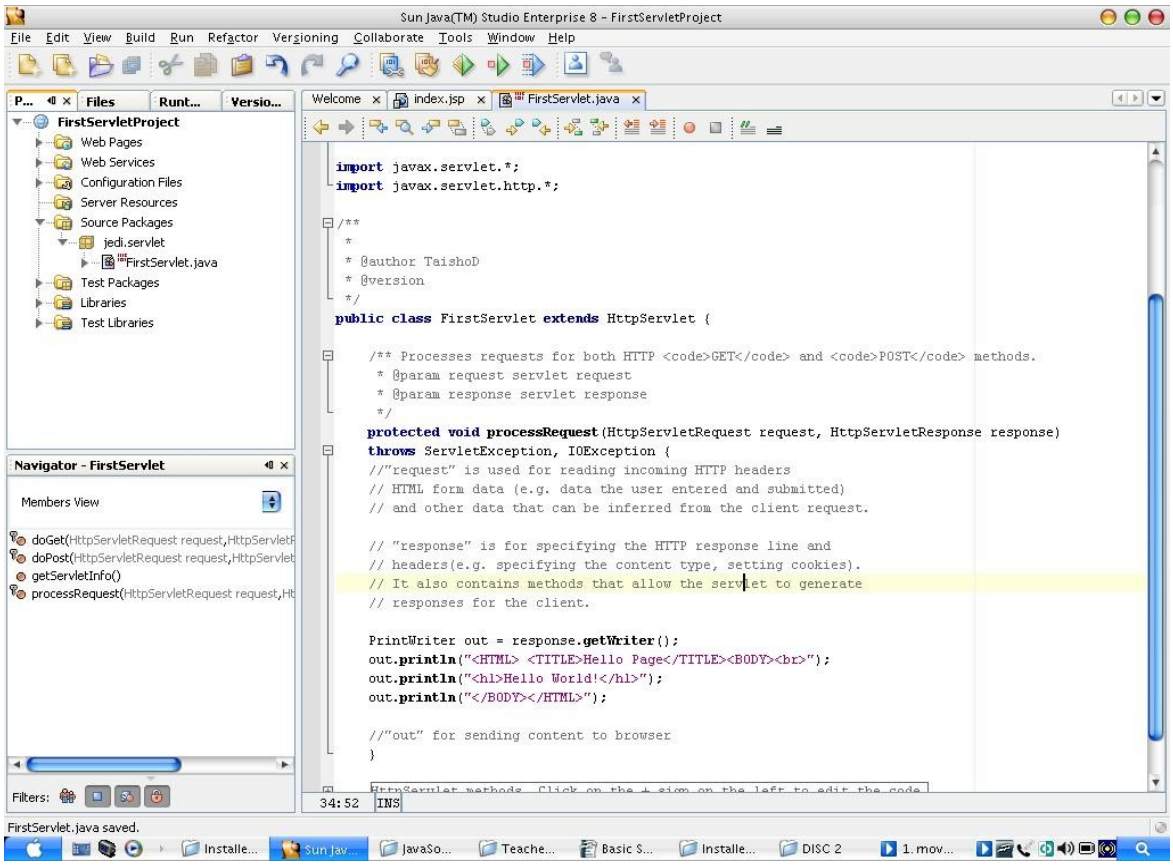

Figura 7: Novo método processRequest

Para executar, pressione **Shift + F6**. A IDE irá então empacotar, instalar e invocar a *Servlet e* automaticamente chamar o navegador WEB.

3. Ciclo de Vida da Classe Servlet

Uma *Servlet* é gerenciada através de um ciclo de vida bem definido descrito nas especificações Servlet. O ciclo de vida do Servlet descreve como ele é carregado, instanciado, inicializado, requisitado, destruído e finalmente coletado (retirado de memória – *garbage collection*). O ciclo de vida de um Servlet é controlado pelo contêiner em que ele é instalado.

O ciclo de vida da *Servlet* permite ao contêiner resolver os problemas de endereçamento e performance do CGI e as preocupações com segurança da API de programação de um servidor baixo-nível. Um contêiner deve poder executar todas as *Servlets* em uma única JVM (Java Virtual Machine – Máquina Virtual Java). Por esta razão, as *Servlets* podem eficientemente compartilhar dados entre si e evitar que uma acesse os dados particulares de outra. As *Servlets* podem também permitir persistência entre requisições como objetos instanciados, utilizando assim menos memória do que processos completamente empacotados.

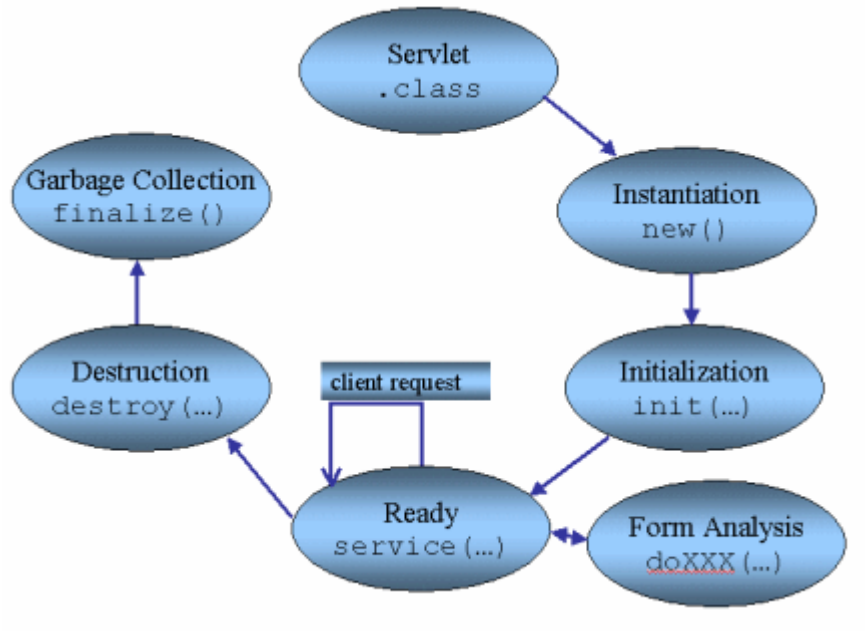

Figura 8: Ciclo de vida da Servlet

A figura mostra os principais eventos na vida de uma *Servlet*. É importante notar que para cada um destes eventos há um método que será invocado pelo contêiner.

3.1. Instanciação

Neste estágio, a classe *Servlet* é carregada para a memória e uma instância é criada pelo contêiner.

Por padrão, um *contêiner* pratica o que chamamos de ***lazy loading*** (carregamento tardio). Utilizando este método, uma classe *Servlet* é carregada para a memória, instanciada e inicializada somente depois que uma requisição for feita. Isto diminui o tempo de inicialização da aplicação, entretanto significa também que haverá um pouco de atraso associado à primeira requisição de cada *Servlet*. Se este comportamento for indesejável, cada *Servlet* deve ser configurada para ser carregada juntamente com a aplicação. Isto será discutido posteriormente quando abordarmos o descritor de carregamento.

Como podemos ver na figura, uma *Servlet* passa pela fase de instanciação somente uma vez durante seu ciclo de vida. Isto significa que o atraso associado ao seu carregamento em memória acontece somente uma vez. Isto mostra a vantagem desta sobre outras tecnologias.

O método relevante que o *contêiner* irá chamar neste estágio será o construtor. Não é recomendado que se coloque qualquer código no construtor. A maioria das funcionalidades que os desenvolvedores querem adicionar aos construtores envolvem definição de objetos ou inicialização

de variáveis. *Servlets* possui uma fase separada para este tipo de atividade.

3.2. Inicialização

Nesta fase, *Servlet* é primordial para o uso na aplicação. Assim como a fase de instanciação, uma *Servlet* passa por este estágio somente uma vez. Isto só ocorre após a fase em que nosso objeto é inicializado ao ser chamado pela *Servlet*.

O método que é chamado pelo contêiner neste ponto é o método **init()**. A assinatura completa do método é apresentada abaixo.

```
public void init(ServletConfig config)
```

Como podemos ver, este método tem um parâmetro: uma instância do objeto *ServletConfig*. Este objeto contém informações sobre a configuração da *Servlet* bem como fornece meios para que a *Servlet* acesse informações da aplicação e seus recursos.

Como mencionado anteriormente, qualquer código de configuração ou inicialização não deve ser colocado na área do construtor, em vez disso, deve ser colocado dentro do método **init**. Se um desenvolvedor implementar este método, é importante que realize uma chamada ao **super.init(config)**. Isto garante que a ação de inicialização padrão da *Servlet* seja executada e que sua configuração seja apropriadamente definida.

Após a inicialização, a *Servlet* estará apta a receber as requisições dos clientes.

Este método somente será chamado novamente quando o servidor recarregar a *Servlet*. O servidor não pode recarregar uma *Servlet* até que esta seja destruída através do método *destroy*.

3.3. Pronta

Esta é a fase quando uma *Servlet* está no auge do seu ciclo de vida. Nesta, a Servlet pode ser chamada repetidamente pelo contêiner para prover sua funcionalidade.

O método chamado pelo contêiner nesta fase é **service()** e possui a seguinte assinatura:

```
public void service(ServletRequest req, ServletResponse res)
```

Os objetos ServletRequest e ServletResponse passados para este método provêem métodos para extrair informação das requisições dos usuários e métodos para gerar as respostas.

Um detalhe importante a ser observado, o contêiner realiza repetidas chamadas ao método **service** usando *threads* separadas. Geralmente, há somente uma instância ativa da *Servlet* consumindo espaço em memória e atendendo às diversas requisições. Esta é outra vantagem que o Servlet Java possui é também uma das principaisrazões porque uma *Servlet* (e o seu método *service*) é projetado para conter um *thread-safe*.

Para Servlets específicos HTTP (Servlets estendendo HttpServlet), os desenvolvedores não devem implementar o método service diretamente. Ao invés disto, devem implementar qualquer um dos seguintes métodos:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
public void doPost(HttpServletRequest req, HttpServletResponse res)
public void doPut(HttpServletRequest req, HttpServletResponse res)
public void doTrace(HttpServletRequest req, HttpServletResponse res)
```

Cada um destes métodos corresponde a um método HTTP específico (GET, POST, ...). O método apropriado é então chamado quando o Servlet recebe a requisição HTTP.

Já que a maioria das chamadas HTTP, que os desenvolvedores devem se preocupar, são dos métodos GET ou POST, *Servlets* podem implementar doGet, doPost ou ambos.

3.4. Destruição

Em algumas ocasiões, quando o contêiner ficar sem memória ou detectar que a quantidade de

memória livre possui pode gerar algum problema, o contêiner tentará liberar memória destruindo uma ou mais instâncias da *Servlet*. Qual será removida é determinado pelo contêiner e o desenvolvedor não tem controle direto.

Um contêiner poderá liberar uma instância da *Servlet* como parte de seu processo de desligamento.

Quando a *Servlet* está para ser removida de um gerenciamento do contêiner, ela está na fase de destruição. O método chamado pelo contêiner, antes de realizar isto, é o **destroy().** Aqui, na nossa Servlet deveria ser codificado para explicitamente liberar os recursos utilizados (ex. Conexões com o Banco de Dados).

3.5. Garbage Collection

Esta fase do ciclo de vida de uma *Servlet* é equivalente a de qualquer outro objeto Java. Relembrando que esta fase ocorre antes que um objeto seja retirado da memória. Os desenvolvedores não têm controle direto quando isso irá ocorrer.

O método do objeto chamado nesta fase é o **finalize()**.

4. Manipulando Requisições e Respostas

O principal objetivo de uma *Servlet* é prover conteúdo dinâmico para o usuário. Por definição, o conteúdo dinâmico muda em resposta a diversas condições. Exemplos dos quais são as requisições específicas de usuário, a hora, temperatura.

Para dar a *Servlet* acesso aos dados de uma requisição, esta é provida em uma instância do objeto *ServletRequest* que esconde estes detalhes. *Servlets* baseados em HTTP possuem uma subclasse, *HTTPServletRequest*, que provê métodos adicionais para obter informação específica do *HTTP*, como informação sobre *cookies*, detalhes de cabeçalhos, entre outros.

4.1. Dados e Parâmetros de Formulário

4.1.1. Método request.getParameter

Um dos cenários que freqüentemente requer mais frequentemente conteúdo dinâmico é quando queremos que nossa aplicação responda aos dados do usuários apresentados em um formulário.

Veja o exemplo seguinte:

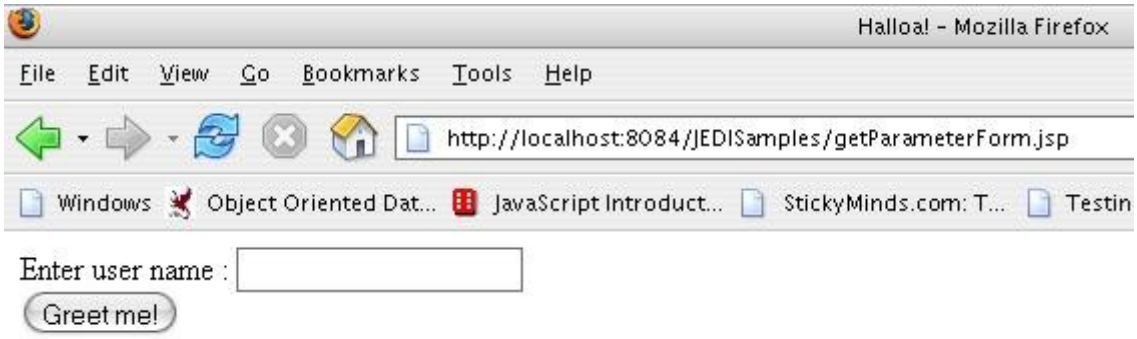

Figura 9: Entrada do Nome do Usuário

Dado o seguinte formulário, mostrado na figura anterior, que obtém o nome do usuário, queremos produzir uma resposta customizada para qualquer nome submetido.

Para este e outros cenários similares, Java provê o método *getParameter* no objeto *HttpServletRequest*. Este método possui a seguinte assinatura:

```
public String getParameter(String parameterName)
```

Este método obtém o nome do parâmetro que desejamos obter e retorna o valor da String que representa este parâmetro.

Abaixo segue um código exemplo que obtém o nome do usuário e dá saída em um simples cumprimento, seguido pelo HTML usado para mostrar o formulário.

```
public class GetParameterServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    // retorna o valor solicitado pelo usuário
    String userName = request.getParameter("userName");
    // retorna um objeto PrintWriter e utiliza-o para a mensagem de saída
    PrintWriter out = response.getWriter();
    out.println("<HTML><BODY><H1>");
    out.println("HELLO AND WELCOME, " + userName + "!");
    out.println("</H1></BODY></HTML>");
    out.close();
  }
}
```

Temos a seguir, o código para HTML exemplo utilizado para mostrar o formulário:

```
<HTML>
```

```
      <TITLE>Halloa!</TITLE>
      <BODY>
         <form action="GetParameterServlet" method="post">
              Enter user name: <input type="text" name="userName"/> <br/>
              <input type="submit" value="Greet me!"/>
          </form>
       </BODY>
</HTML>
```

Ao entrar com o valor "JEDI" no formulário, obtemos o seguinte resultado:

Figure 10: Saída do GetParameterServlet

4.1.2. Método request.getParameterValues

Existem momento que necessitamos recuperar dados de um formulário com múltiplos elementos com o mesmo nome. Neste caso, usando apenas o método *getParameter* descrito anteriormente, será retornado apenas o valor do primeiro elemento com aquele nome. Para recuperar todos os valores, usamos o método *getParameterValues*:

```
public String[] getParameterValues(String parameterName)
```

Este método é similar ao que acabamos de discutir. Também recebe o nome do parâmetro que queremos recuperar o valor. Desta vez, ao invés de uma única String, retorna um *array* de Strings.

Este é um exemplo:

```
public class GetParameterValuesServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletExceptionm IOException{
    String paramValues[] = request.getParameterValues("sports");
    StringBuffer myResponse = new StringBuffer();

    PrintWriter out = response.getWriter();
    out.println("<HTML><TITLE>Your choices</TITLE>");
    out.println("<BODY><H1>Your choices were : </H1>");

    for (int i = 0; i < paramValues.length; i++) {
      out.println("<br/><li>");
      out.println(paramValues[i]);
    }
    out.println("</BODY></HTML>");
  }
}
```

A seguir, temos o código para o HTML usado para mostrar o formulário:

```
<html>
  <title>Choice selection</title>
  <body>
    <H1>What sports activities do you perform?</h1>
    <form action="GetParameterValuesServlet" method="get">
      <input type="checkbox" name="sports" value="Biking"> Biking
```

```
        <br/>
        <input type="checkbox" name="sports" value="Table Tennis"> Table Tennis
        <br/>
        <input type="checkbox" name="sports" value="Swimming"> Swimming
        <br/>
        <input type="checkbox" name="sports" value="Basketball"> Basketball
        <br/>
        <input type="checkbox" name="sports" value="Others"> Others
        <br/>
        <input type="submit">
      </form>
    </body>
  </html>
```

Figure 11: Recuperando dados de um formulário com múltiplos elementos

Se as opções "Biking", "Table Tennis", e "Swimming" fossem escolhidas, a saída seria:

Figure 12: Saída do GetParameterValuesServlet

4.1.3. request.getParameterNames

Algumas vezes necessitamos que a *Servlet* conheça o nome de um ou mais parâmetros do formulário sem ter que codificá-los dentro do *Servlet*. Neste caso, podemos usar o método

getParameterNames.

```
public Enumeration getParameterNames()
```

O objeto *Enumeration* que este método retorna contém todos os nomes de parâmetros contidos nesta requisição do usuário.

4.2. Recuperando a informação da URL de Requisição

Relembrando, uma requisição HTTP é composta pela seguintes partes:

```
http://[host]:[porta]/[caminhoDaRequisição]?[queryString]
```

Podemos recuperar os valores atuais de cada uma destas partes da requisição do usuário chamando alguns métodos do objeto *HttpServletRequest*.

- **Host** – request.getServerName()
- **Porta** – request.getServerPort()
- **Caminho da Requisição – em** Java, o caminho da requisição é dividida em 2 componentes lógicos :
  - **Contexto –** o contexto de uma aplicação WEB. Pode ser recuperado chamando request.getContextPath()
  - **Informação do caminho –** o resto de uma requisição depois do nome do contexto. Pode ser recuperado chamando request.getPathInfo()
- **Query String –** request.getQueryString()

Então, por exemplo, com uma URL de requisição:

```
http://www.myjedi.net:8080/HelloApp/greetUser?name=Jedi
```

Se chamarmos os métodos mencionados, os seguintes resultados serão exibidos:

`request.getServerName()`	`www.myjedi.net`
`request.getServerPort()`	`8080`
`request.getContextPath()`	`HelloApp`
`request.getPathInfo()`	`greetUser`
`request.getQueryString()`	`name=Jedi`

4.3. Informação de Cabeçalho

Informação do cabeçalho pode ser recuperada pela *Servlet* chamando os seguintes métodos em *HttpServletRequest*:

- **getHeader(String nome)** – retorna o valor do cabeçalho especificado como um String. Se o cabeçalho especificado não existir, este método retorna null.
- **getHeaders(String nome) –** similar ao *getHeader*(). Entretanto, ele recupera todos os valores do cabeçalho especificado como um objeto *Enumeration*.
- **getHeaderNames() -** este método retorna os nomes de todos os cabeçalhos incluídos na requisição HTTP como um objeto *Enumeration*.
- **getIntHeader(String nome) –** retorna o valor de um cabeçalho especificado como um int. Se o cabeçalho especificado não existir na requisição, o método retorna -1.
- **getDateHeader(String nome) –** retorna o valor de um cabeçalho especificado como um valor long que representa um objeto Date. Se o cabeçalho especificado não existir, este método retorna -1. Se o cabeçalho não puder ser convertido em um objeto do tipo Date, este método lança uma *IllegalArgumentException*.

4.4. Geração da Saída

Em todos os exemplos anteriores, conseguimos gerar saídas dinâmicas para o usuário. Fizemos isto usando métodos do objeto *HttpServletResponse*.

Até agora, usamos o método *getWriter*. Este método retorna um objeto PrintWriter associado com nossa resposta para o usuário. Para ajudar a colocar as coisas na perspectiva correta, devemos lembrar que o objeto System.out que usamos regularmente para dar saída no conteúdo para o console **também** é uma instância do objeto PrintWriter. Ou seja, eles se comportam quase da mesma maneira: se você tiver uma instância do objeto PrintWriter, simplesmente chame os métodos ***print*** ou ***println*** para gerar a saída.

O código da nossa primeira classe *Servlet* será visto a seguir com o código de saída em negrito.

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    // objeto "out" é utilizado para enviar o conteúdo para o browser
    PrintWriter out = response.getWriter();
    out.println("<HTML> <TITLE>Hello Page</TITLE><BODY><br>");
    out.println("<h1>Hello World!</h1>");
    out.println("</BODY></HTML>");
  }
}
```

Outros métodos importantes no objeto *HttpServletResponse* são:

- **setContentType** – informa ao navegador do cliente o tipo MIME da saída que ele irá receber. Todo o conteúdo que geramos até agora foi HMTL. Poderíamos facilmente enviar outros tipo de conteúdo como JPEG, PDF, DOC, XLS, etc. Para conteúdo que não é texto, os métodos print do objeto PrintWriter que estamos usando até agora são insuficientes. Para gerar saída não textual, fazemos uso de outro método.
- **getOutputStream** – este método recupera uma instância do objeto OutputStream associado com nossa resposta para o usuário. Com o OutputStream, podemos usar os objetos e métodos padrão de E/S Java para produzir todos os tipos de saída.

Abaixo segue um código exemplo que irá gerar um arquivo JPG contido em uma aplicação web para o usuário.

```
public class JPEGOutputServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response) {
    // definir um array de byte para armazenamento dos dados
    byte bufferArray[] = new byte[1024];

    // retornar o ServletContext que será utilizado para o retorno
    ServletContext ctxt = getServletContext();

    // informar ao browser que está enviando uma imagem
    response.setContentType("image/gif");

    // criar o objeto de saída em stream que será produzida
    ServletOutputStream os = response.getOutputStream();

    // criar o objeto de recurso para o input stream
    InputStream is = ctxt.getResource("/WEB-INF/images/logo.gif").openStream();

    // ler o conteúdo do recurso de escrita e gerar a saída
    int read = is.read(bufferArray);
    while (read != -1) {
      os.write(bufferArray);
```

```
        read = is.read(bufferArray);
      }
      // fechar os objetos utilizados
      is.close();
      os.close();
    }
  }
```

5. Configuração, Empacotamento e Distribuição

Em todos os exemplos realizados, usamos as ferramentas da IDE Sun Enterprise Studio 8 para facilitar os detalhes da configuração, do empacotamento e da instalação da aplicação WEB. Iremos agora, aprofundar estes detalhes.

5.1. Configuração da Aplicação WEB

A especificação *Servlet*, define um arquivo XML chamado **web.xml** que atua como um arquivo de configuração para as aplicações WEB. Este arquivo é chamado de descritor de distribuição.

```
<web-app version="2.4"
 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
 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>jedi.Servlet.FirstServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/FirstServlet</url-pattern>
 </servlet-mapping>
 <session-config>
  <session-timeout>
   30
  </session-timeout>
 </session-config>
 <welcome-file-list>
  <welcome-file>
   index.jsp
  </welcome-file>
 </welcome-file-list>
</web-app>
```

Temos acima um exemplo do arquivo **web.xml** utilizado no projeto *FirstServlet*. Usaremos como ponto de partida na exploração deste arquivo.

NOTA: O arquivo web.xml da aplicação WEB construída pela IDE, pode ser encontrado expandindo a aba de Arquivos de Configuração na **View -> Project**.

<web-app>

Esta linha é utilizada como elemento raiz do arquivo de configuração e como uma declaração das informações necessárias (até seus atributos), para o contêiner poder reconhecer o arquivo como um arquivo descritor válido.

<servlet>

Cada instância deste elemento define uma *Servlet* para ser usada pela aplicação. Possui os seguintes *nodes* filhos:

- **<servlet-name>** - Um nome lógico fornecido pelo desenvolvedor que será usado para todas as referências futuras para a Servlet.
- **<servlet-class>** - O nome da classe *Servlet* completamente qualificado.
- **<load-on-startup>** – Opcional. Tendo uma entrada e valor para este elemento, informa ao contêiner que a *Servlet* deve ser construída e inicializada durante o início do contêiner ou aplicação, ultrapassando o lento carregamento do código. O valor para este elemento é um número que dita o ordem de carregamento comparando ao <load-on-startup> de outras Servlets.

<servlet-mapping>

Cada instância deste elemento define um mapeamento para uma *Servlet*. Possui os seguintes *nodes* filhos:

- **<servlet-name>** - O nome lógico da *Servlet* para ser mapeado. Deve ser definido previamente no arquivo descritor.
- **<url-pattern>** - A URL utilizada para que esta *Servlet* seja mapeada. O valor **/***, fará com que todas as solicitações da aplicação seja redirecionada para seu *Servlet*. No exemplo dado, a URL é **/FirstServlet**. Isto significa que todas as solicitações de URL para http://[host]:[port]/FirstServletProject/FirstServlet será manipulado por FirstServlet.

Lembre-se que todas as definições de *Servlet* devem ser fornecidas antes de adicionar quaisquer Servlet-mappings.

<session-config>

Este elemento define os detalhes do gerenciamento da configuração da seção.

<welcome-file-list>

Este elemento define um componente WEB que será automaticamente carregado se o usuário entrar com uma solicitação para o servidor sem especificar um recurso em particular. Por exemplo, uma solicitação para *http://[host]:[port]/FirstServletProject* originará o arquivo aqui definido para ser carregado.

Mais de um arquivo pode ser especificado nesta lista. Somente o primeiro arquivo visível para o contêiner WEB será carregado.

5.2. Empacotando a Aplicação da WEB

Observaremos novamente a estrutura de uma aplicação WEB como indicada pela especificação *Servlet*:

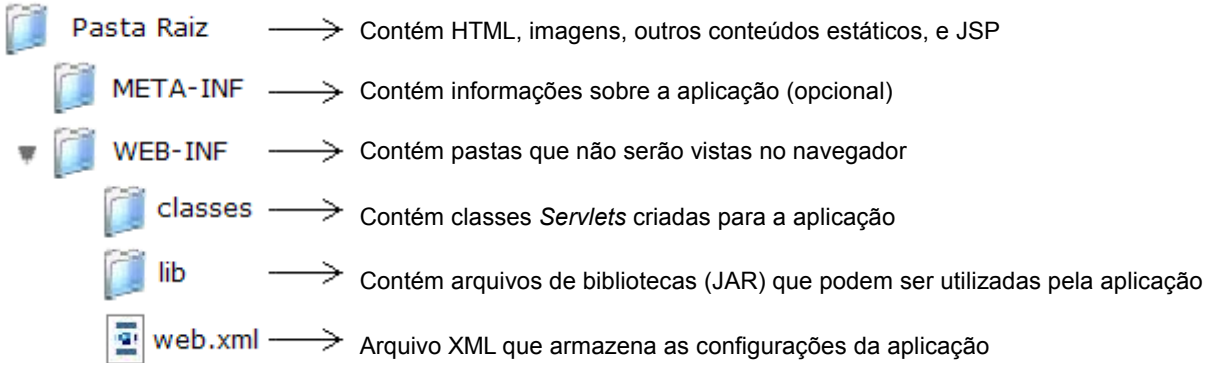

Figura 13: Estrutura do Diretório de uma aplicação Java WEB

A aplicação pode ser instalada em um servidor fazendo uso de um arquivo de WAR (**W**EB **AR**chive). Arquivos WAR são o mesmo que JAR (**J**ava **AR**chive); contêm o código de aplicação comprimido utilizando o formato ZIP.

5.2.1. Gerando arquivos WAR a partir de projetos existentes no *Sun Studio Enterprise*

É muito simples produzir o arquivo WAR contendo nossa aplicação web a partir de um projeto existente no **Sun Studio Enterprise 8**. Basta pressionar o botão direito do mouse no nome do projeto em ***Project view***, e selecionar ***Build Project***. A IDE então procederá o empacotamento da aplicação.

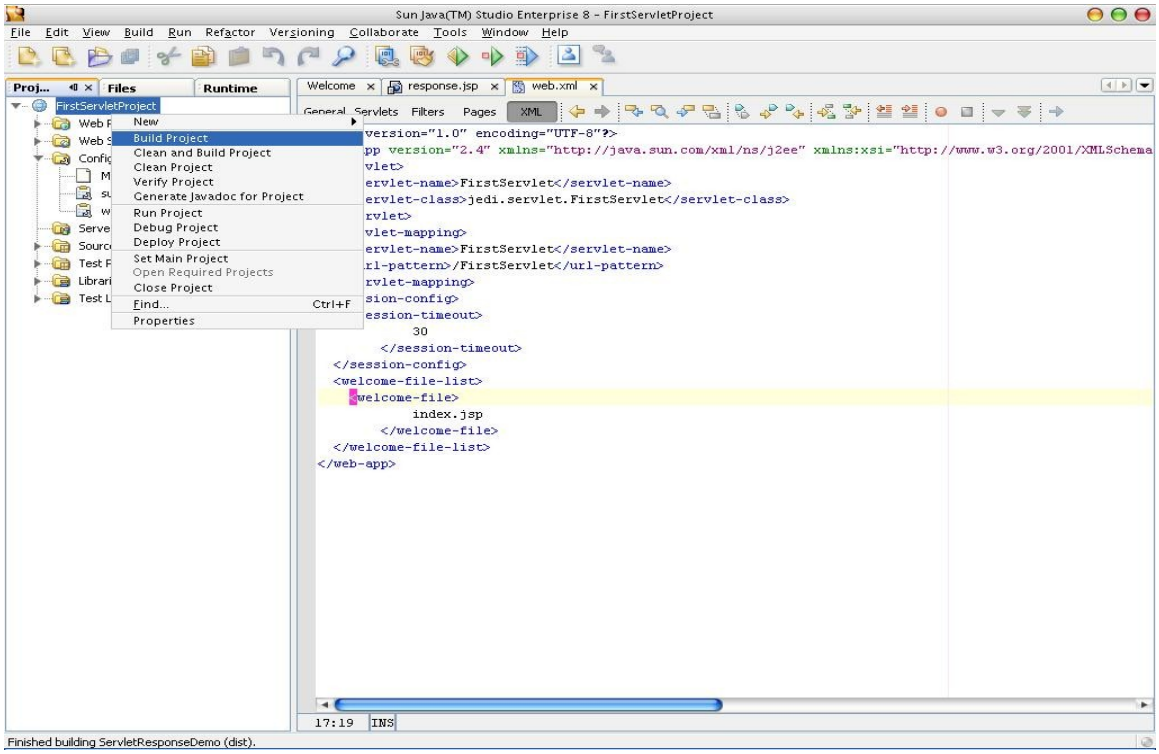

Figura 14: Empacotando o projeto

A IDE informará se a operação de **construção** foi bem sucedida, e o local onde o arquivo WAR foi criado.

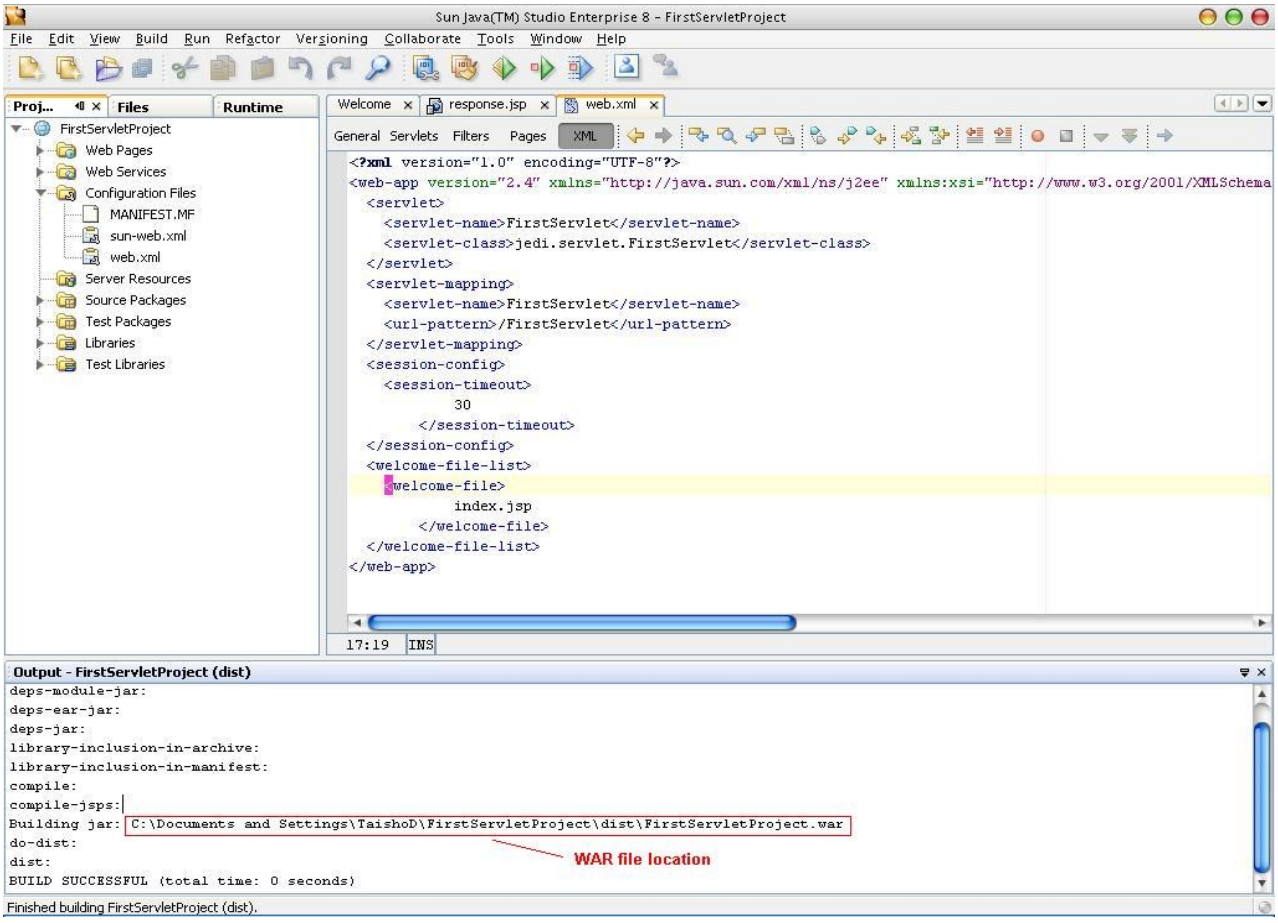

Figura 15: Construção com sucesso

5.2.2. Usando um arquivo de construção Ant para empacotar a aplicação

Além de usar uma IDE para empacotar a aplicação da WEB, podemos também fazer uso de uma ferramenta de construção para automatizar a compilação e o processo de empacotamento.

Uma ferramenta de construção que possui uma ampla utilização é chamada **Apache Ant**. É um projeto de código aberto da *Apache Software Foundation*, e seu download pode ser realizado no endereço *http://ant.apache.org*.

Basicamente, **Ant** lê em um arquivo de construção (chamado **build.xml**). Este arquivo de construção é composto de *targets* (alvos), as quais essencialmente definem atividades lógicas que podem ser executadas pelo arquivo de construção. Estes targets são por sua vez, compostos de uma ou mais tarefas que definem os detalhes de como os targets realizam suas atividades.

Requisitos do arquivo de construção:

- Deve ser locado na **Raiz do Projeto** de acordo com a estrutura do diretório recomendado pela *Apache Software Foundation* para desenvolvimento de aplicação WEB.
- Adicionalmente, deve existir também um diretório **lib** na **Raiz do Projeto** que conterá todas dependências JAR da aplicação.
- Deve existir um arquivo chamado **build.properties** no mesmo diretório que o roteiro de construção e deve conter valores para as seguintes propriedades:
  - **app.name** – nome da aplicação / projeto
  - **appserver.home** – o diretório de instalação da instância Sun Application Server 8.1

Segue a estrutura de diretório recomendado para desenvolvimento de aplicação web:

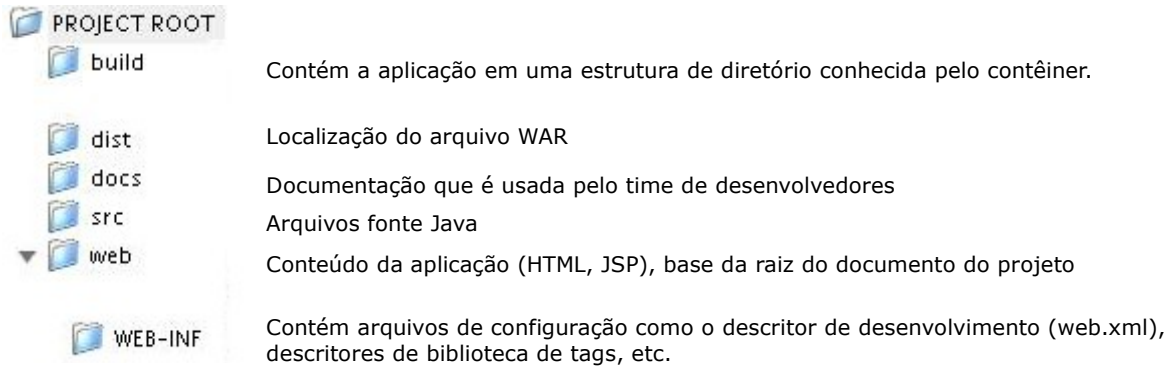

Figura 16: Estrutura de diretório recomendada para o desenvolvimento de aplicação WEB

Esta estrutura de diretório foi projetada para ser distinta da estrutura de diretório exigido pela especificação *Servlet*. A Apache recomenda por possuir as seguintes vantagens:

- O conteúdo de diretório de fonte são mais facilmente administrados, deslocado, ou aprovado se a versão de desenvolvimento não misturada.
- O controle de código de fonte é mais fácil de administrar em diretórios que contenham só arquivos de fonte (nenhuma classe compilada, etc.).
- Os arquivos que compõem a distribuição da aplicação é mais fácil selecionar quando a hierarquia de desenvolvimento for separada.

Pode ser difícil compreender à primeira vista. Para ajudar a compreensão, todo os requisitos para compilar e empacotar nosso exemplo *FirstServlet* é fornecido na pasta *samples/FirstServlet*.

Para realizar o empacotamento de uma aplicação usando esta estrutura, execute a seguinte instrução (no mesmo diretório contendo o arquivo de construção).

```
ant dist
```

Esta instrução chamará o *target* **dist** no arquivo de construção, o qual gerará o arquivo WAR e o colocará no diretório *dist*. Este arquivo WAR pode, então, ser carregado no contêiner usando as ferramentas administrativas oferecidas pelo contêiner.

5.3. Desenvolvimento em Servidor

Os contêiners Servlet geralmente contêm ferramentas administrativas que podem ser usadas para desenvolver aplicações WEB. Neste momento, veremos os passos exigidos para desenvolver o arquivo WAR gerado no aplicativo Sun Application Server 8.1.

- Primeiro passo, execute o *log-in* no console administrativo. Isto pode ser acessado através da seguinte URL em seu navegador: *http://localhost:[ADMIN_PORT]* onde ADMIN_PORT é a porta configurada durante a instalação para manipular os assuntos administrativos.

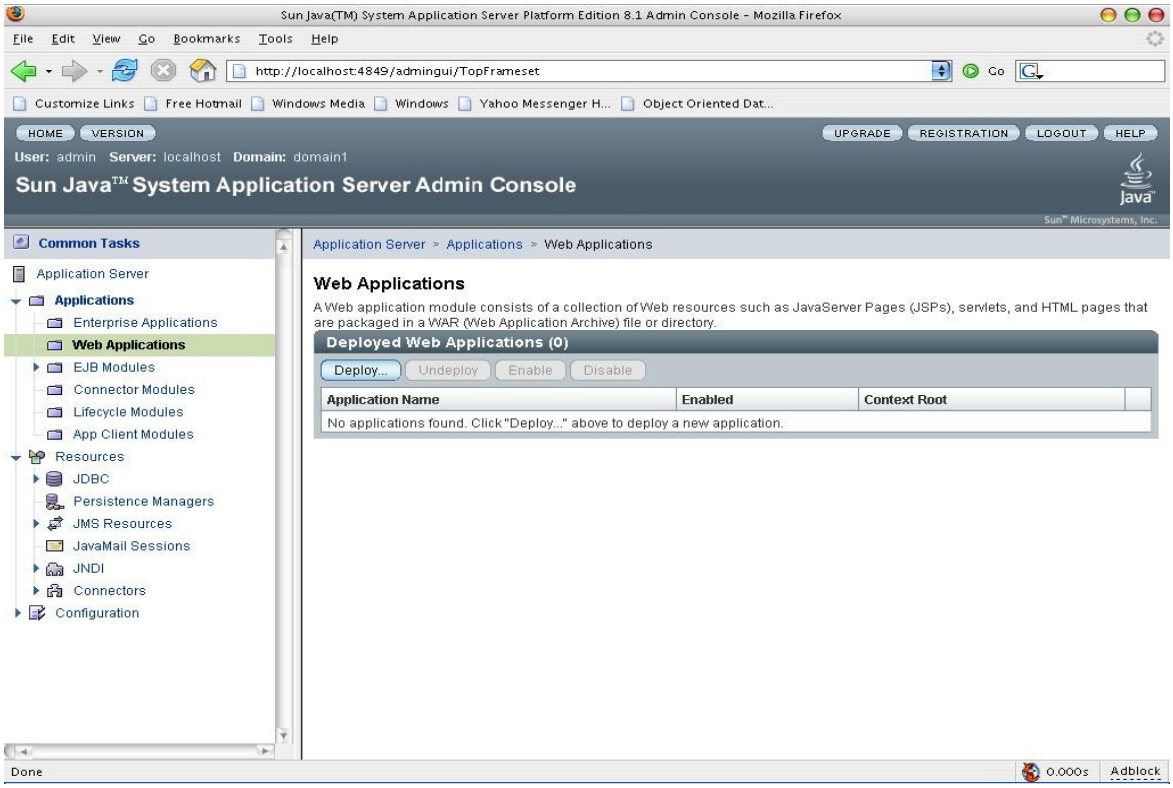

Figure 17: Instalando um arquivo WAR

- Segundo, selecione a aba **Web Aplications**, no painel à esquerda
- Pressione o botão **Deploy** encontrado no painel à direita
- Na tela seguinte, pressione o botão **Browse** para selecionar o arquivo WAR para instalar
- Pressione o botão **Next** encontrado na parte superior à direita
- Pressione o botão de **Finish** na próxima tela
- Parabéns, sua aplicação foi instalada

6. Servlet e Parâmetros de Aplicação

6.1. ServletConfig e Parâmetros de Inicialização da Servlet

O objeto da classe *ServletConfig* é passado para uma *Servlet* específica durante sua fase de inicialização. Usando isto, uma *Servlet* pode recuperar informações específicas para si mesmo (parâmetros de inicialização). A *Servlet* também pode ganhar acesso a uma instância do objeto de *ServletContext* usando o objeto da classe *ServletConfig*.

Os parâmetros de inicialização são de grande uso, especialmente quando lidamos com informações que podem variar com cada desenvolvimento da aplicação. Além disso, fornecendo dados para a *Servlet* como parâmetros, evitamos a codificação desses parâmetros diretamente na *Servlet*, permite aos desenvolvedores a habilidade de mudar o comportamento da *Servlet* sem ter que recompilar o código.

Podemos adicionar parâmetros de inicialização para a *Servlet* especificando-os na definição do *deployment descriptor* do Servlet. A seguir, temos uma amostra:

```
...
<Servlet>
 <Servlet-name>FirstServlet</Servlet-name>
 <Servlet-class>jedi.Servlet.FirstServlet</Servlet-class>
 <init-param>
  <param-name>debugEnabled</param-name>
  <param-value>true</param-value>
 </init-param>
</Servlet>
...
```

As tags **<init-param>** e **</init-param>** informam ao contêiner que estamos começando e terminando a definição de parâmetro, respectivamente. <param-nome> define o nome do parâmetro, e **<param-value>** define seu valor.

Para ter acesso aos parâmetros da *Servlet*, devemos primeiro obter o acesso ao seu objeto *ServletConfig*, que pode ser feito recuperado solicitando ao método *getServletConfig*(). Após isso, o valor de parâmetro poderá ser recuperado através de uma String chamando o método *getInitParameter*() e fornecendo o valor de **<param-nome>** como o parâmetro.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  ServletConfig config = getServletConfig();
  String isDebugEnabled = config.getInitParameter("debugEnabled");
  if (isDebugEnabled.equals("true") {
    ...
}
```

A amostra de código acima ilustra o procedimento.

6.2. ServletContext e Parâmetros de Aplicação

O objeto de *ServletContext* informa a *Servlet* o acesso ao contexto de aplicação.

Pense sobre o contexto de aplicação como a área na qual as aplicações se movem. Esta área é fornecida pelo contêiner para cada aplicação WEB. Com cada contexto da aplicação sendo diferente uma do outra, uma aplicação não pode acessar o contexto de outra aplicação.

Ter acesso a este contexto é importante, porque através deste, a *Servlet* pode recuperar parâmetros da aplicação. É possível também armazenar dados que podem ser recuperados por quaisquer componentes na aplicação.

Do mesmo modo que parâmetros de inicialização podem ser fornecidos para *Servlets* individuais, eles também podem ser fornecidos para uso pela aplicação inteira.

```
<context-param>
```

```
 <param-name>databaseURL</param-name>
 <param-value>jdbc:postgresql://localhost:5432/jedidb</param-value>
</context-param>
```

O código acima exemplifica como adicionar diferentes parâmetros de aplicação. O elemento XML usado aqui é **<context-param>**. Cada instância de tal elemento define um parâmetro para uso pela aplicação inteira. **<param-name>** e **<param-value>** trabalham da mesma maneira que a tag **<init-param>**.

NOTA: Novamente, lembre-se que a especificação é restrita quanto a ordenação de seus elementos dentro do descritor de desenvolvimento. Para manter o arquivo **web.xml** válido, todas as entradas **<context-param>** devem ser localizadas ANTES de quaisquer entradas <Servlet>.

O procedimento para recuperar os valores dos parâmetros é bem parecido com o utilizado para recuperar os parâmetros de *Servlets* específicas. É uma instância de *ServletContext* que a *Servlet* deve manipular. Isto pode ser recuperado chamando o método *getServletContext*() de uma instância do objeto *ServletConfig* da *Servlet*.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
  ServletContext ctxt = getServletConfig().getServletContext();
  String jdbcURL = ctxt.getInitParameter("databaseURL");
  // código customizado
  setURL(jdbcURL);
  ...
}
```

6.3. Resumo

- A tecnologia *Servlets* são a solução Java para a produção de conteúdo dinâmico para a WEB em resposta a tecnologia CGI.
- *Servlets* possui várias vantagens sobre CGI, incluindo uma reduzida na área de ocupação de memória e menos despesa para cada solicitação.
- Uma *Servlet* está completamente administrada pelo seu contêiner. O único código necessário para os desenvolvedores é a funcionalidade da implementação.
- Para criar *Servlets*, os desenvolvedores devem criar subdivisões de classe de *HttpServlet* e seus lugares de implementações funcionais em quaisquer dos métodos, **doGet** ou **doPost**.
- Detalhes de pedido podem ser recuperados por *HttpServletRequest*, e métodos geradores de resposta podem ser acessados por *HttpServletResponse*. Estes são passados como parâmetros para **doGet** e **doPost**.
- Para instalar *Servlets* no contêiner WEB, é necessário criar arquivos WAR. Este processo de empacotamento pode ser automatizado por qualquer IDE ou pelo uso de uma ferramenta de construção.
- Descritores do desenvolvimento são partes essenciais da aplicação. Eles devem ser localizados no diretório de aplicação **WEB-INF** e seguir determinadas regras.
- Parâmetros podem ser fornecidos pela aplicação usando o descritor de desenvolvimento e recuperado usando os métodos em instâncias *ServletConfig* ou *ServletContext*.

7. Exercícios

7.1. Perguntas sobre o Ciclo de Vida da Servlet

Responda às seguintes perguntas sobre as fases da *Servlet*:

1. Quais são as 5 fases do ciclo de vida de uma *Servlet*?
2. Quais são os métodos associados a cada fase do ciclo de vida de uma *Servlet*?
3. Por que o código colocado no método *service* de uma *Servlet* precisa ser thread-safe?
4. Quando uma *Servlet* é destruída?
5. Quantas vezes o código colocado dentro do método *init*() do *Servlet* pode ser chamado? E o código dentro do método *service*()?
6. Caso a *Servlet* estenda a classe *HttpServlet*, quais métodos adicionais poderemos implementar para produzir a funcionalidade necessária?

7.2. Exercícios de Manipulação de Requisição/Geração de Resposta

1. Dado o seguinte formulário:

```
<HTML>
 <BODY>
  <form action="AddServlet" method="post">
    Enter number 1 : <input type="text" name="operand1"/> </br>
    Enter number 2 : <input type="text" name="operand2"/> </br>
    <input type="submit" value="Perform addition"/>
   </form>
  </BODY>
 </HTML>
```

Crie uma *Servlet* chamado *AddServlet* que irá recuperar 2 números dados pelo usuário, adicione-os, e gere o resultado.

2. Dado o seguinte formulário:

```
<html>
 <title>Menu</title>
 <body>
  <H1>Which food items do you want to order?</h1>
  <form action="MenuSelectionServlet" method="post">
   <table>
    <tr>
     <td><input type="checkbox" name="order" value="20"> Sundae </td>
     <td> P 20 </td>
    </tr><tr>
     <td><input type="checkbox" name="order" value="25"> Reg. Burger </td>
     <td> P 25 </td>
    </tr><tr>
     <td><input type="checkbox" name="order" value="15"> Dessert Pie </td>
     <td> P 15 </td>
    </tr><tr>
     <td><input type="checkbox" name="order" value="70"> Rice Meal </td>
     <td> P 70 </td>
    </tr><tr>
     <td><input type="submit"></td>
    </tr>
   </table>
  </form>
 </body>
</html>
```

Crie uma *Servlet* chamado *MenuSelectionServlet* que irá recuperar as seleções feitas pelo usuário, adicionar os seus valores, e retornar o resultado computado para o usuário.

7.3. Perguntas sobre implementação

1. Identifique o elemento no arquivo **web.xml** que será responsável por:

   a) Conter o nome lógico usado para se referir a uma *Servlet*.
   b) Ser o elemento root do arquivo de configuração.
   c) Definir um mapeamento entre uma *Servlet* e uma solicitação do usurário.

2. Reorganize as seguintes informações no ordem correta de seu aparecimento dentro de um arquivo XML:

   session-config
   servlet
   servlet-mapping
   welcome-file-list

3. Suponha termos uma *Servlet* cujo nome é *TrialServlet*, crie um mapeamento de modo que *TrialServlet* será chamada para cada pedido:

   http://[host]/[context]/myExerciseServlet.

4. O que são arquivos WAR?
5. Dado um projeto WEB existente em nossa IDE, como um arquivo WAR pode ser gerado?

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.