

INTEGRATION

MAIS DINAMISMO
NA INTERFACE

COM O USUÁRIO

ALEXANDRE JESUS



02

LISTA DE FIGURAS

Figura 1 – Sistema realizando <i>Request/Response</i>	6
Figura 2 – Narração de um jogo de futebol	6
Figura 3 – Exemplo de requisição a uma <i>servlet</i>	8
Figura 4 – Ciclo de vida de uma <i>servlet</i>	9
Figura 5 – Sequência de eventos em uma <i>Servlet</i>	11
Figura 6 – Exemplo de uma URL para requisição.....	15
Figura 7 – Métodos para recuperar cada informação da requisição	15
Figura 8 – Exemplo de formulário HTML.....	17
Figura 9 – Página de resposta de sucesso	21
Figura 10 – Página de resposta de erro	21
Figura 11 – Visão geral do processamento da <i>request</i> pela <i>servlet</i>	22
Figura 12 – Resultado da execução do código do formulário HTML.....	23
Figura 13 – Exemplo do formulário HTML (calculadora) enviando as informações ..	26
Figura 14 – Página resposta.jsp após ter recebido o atributo enviado pela <i>servlet</i> ...	26
Figura 15 – Visão geral do fluxo de processamento da <i>request</i>	27
Figura 16 – Link apontando para a <i>servlet</i> com o parâmetro	29
Figura 17 – Página resposta.jsp apresentando o resultado em HTML ao cliente	31
Figura 18 – Diagrama ilustrando o <i>request</i> gerado através de um link	31
Figura 19 – Criando um projeto web no eclipse – Parte 1	32
Figura 20 – Criando um projeto web no eclipse – Parte 2.....	33
Figura 21 – Configurando o servidor no projeto – Parte 1	33
Figura 22 – Configurando o servidor no projeto – Parte 2.....	34
Figura 23 – Visão geral da primeira configuração do projeto web.....	35
Figura 24 – Criando um projeto web no eclipse – Parte 3.....	35
Figura 25 – Criando uma página HTML – Parte 1	36
Figura 26 – Criando uma página HTML – Parte 2.....	37
Figura 27 – Criando uma classe <i>Servlet</i> – Parte 1	38
Figura 28 – Criando uma classe <i>Servlet</i> – Parte 2.....	39
Figura 29 – Criando uma página JSP– Parte 1	41
Figura 30 – Criando uma página JSP– Parte 2.....	42
Figura 31 – Executando a aplicação no Tomcat (servidor) – Parte 1	43
Figura 32 – Executando a aplicação no Tomcat (servidor) – Parte 2.....	44
Figura 33 – Resultado da execução da página index.html no servidor	44
Figura 34 – Configuração do browser em que será aberta a página.....	45
Figura 35 – Preenchimento do formulário	45
Figura 36 – Resultado do envio dos dados do formulário	46
Figura 37 – Exibição do link que foi adicionado à página.....	46
Figura 38 – Resultado da ação de clicar no link.....	49

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de uma <i>Servlet</i>	10
Código-fonte 2 – Exemplo do método <code>getHeaderNames()</code>	11
Código-fonte 3 – Exemplo do método <code>getHeader()</code>	12
Código-fonte 4 – Exemplo do método <code>setAttribute()</code> e <code>getAttribute()</code>	13
Código-fonte 5 – Exemplo do método <code>getAttributeNames()</code>	13
Código-fonte 6 – Exemplo do método <code>getCookies()</code>	14
Código-fonte 7 – Exemplo do método <code>getSession()</code>	14
Código-fonte 8 – Exemplo do método <code>getParameter()</code>	14
Código-fonte 9 – Exemplo de método <code>addCookie()</code>	16
Código-fonte 10 – Exemplo de método <code>setContentType()</code>	16
Código-fonte 11 – Exemplo de método <code>getWriter()</code>	16
Código-fonte 12 – Exemplo de implementação do formulário HTML	18
Código-fonte 13 – Propriedade <code>action</code> da tag <code><form></code> do HTML	18
Código-fonte 14 – Exemplo de uma <i>servlet</i> com seu mapeamento inicial	19
Código-fonte 15 – Exemplo de uma <i>servlet</i> com seu mapeamento alterado	19
Código-fonte 16 – Propriedade <code>action</code> com o valor ajustado para o mapeamento da <i>servlet</i>	19
Código-fonte 17 – Exemplo de formulário HTML para recuperação de parâmetros	20
Código-fonte 18 – Exemplo de <i>Servlet</i> para recuperação de parâmetros	20
Código-fonte 19 – Exemplo de formulário HTML (calculadora) para a passagem de parâmetros	23
Código-fonte 20 – Exemplo do método <code>doPost</code> da <i>servlet</i> recebendo os parâmetros do formulário	24
Código-fonte 21 – Exemplo da página resposta.jsp com a EL	25
Código-fonte 22 – Exemplo de link apontando para nossa <i>servlet</i>	28
Código-fonte 23 – Exemplo de link apontando para nossa <i>servlet</i> e passando um parâmetro	28
Código-fonte 24 – Exemplo de link apontando para nossa <i>servlet</i> e passando um parâmetro	29
Código-fonte 25 – Exemplo: código na <i>servlet</i> recebendo/repassando o parâmetro enviado por link	30
Código-fonte 26 – Código da página resposta.jsp recuperando atributo por EL passada pela <i>servlet</i>	30
Código-fonte 27 – Formulário de cadastro de produto	38
Código-fonte 28 – Implementação do <i>ProdutoServlet</i>	41
Código-fonte 29 – Página que exibe as informações enviadas pelo formulário	43
Código-fonte 30 – Link para chamar uma <i>Servlet</i>	46
Código-fonte 31 – Método <code>doGet</code> da <i>Servlet</i>	48
Código-fonte 32 – Formulário de cadastro de produto	49

SUMÁRIO

1 MAIS DINAMISMO NA INTERFACE COM O USUÁRIO	5
1.1 Introdução: <i>Servlets</i>	5
1.2 Requisição e resposta	5
1.3 Requisição a uma <i>Servlet</i>	8
1.4 Ciclo de vida de uma <i>servlet</i>	8
1.5 Sequência de eventos no <i>HttpServlet</i>	10
1.5.1 Interface <i>HttpServletRequest</i>	11
1.5.2 Interface <i>HttpServletResponse</i>	16
2 APLICAÇÃO WEB	17
2.1 Recuperação de parâmetros	17
2.2 Mapeamento das <i>Servlets</i>	18
2.3 <i>Servlet request</i>	22
2.4 <i>Request</i> através de links	27
2.5 Prática!	32
REFERÊNCIAS	50

1 MAIS DINAMISMO NA INTERFACE COM O USUÁRIO

1.1 Introdução: *Servlets*

Já evoluímos bastante com o conteúdo e estamos próximos de conseguir desenvolver uma aplicação por completo. Falamos sobre a linguagem Java e como utilizá-la para acessar o banco de dados Oracle. Trabalhamos também com as páginas HTML e as outras tecnologias envolvidas como CSS, Javascript, JQuery e Bootstrap. Chegou o momento de integrar as páginas web com a plataforma Java. Dessa forma, será possível interagir com nossas páginas HTML para realizar operações com o banco de dados, como um cadastro, busca ou alteração, por exemplo. Assim, poderemos “dar vida” a nossas páginas web e implementar, de fato, as funcionalidades do sistema Fintech.

O componente da plataforma Java Web que será responsável por fazer “esse meio de campo” entre a página (interface do usuário) e o modelo de negócio (*model*) é a *servlet*.

Servlets são classes Java que realizam o direcionamento de requisições HTTP feitas por clientes, como os navegadores (Mozilla Firefox[®], Chrome[®]), eles são responsáveis por receber os dados para serem processados e devolver uma resposta a esses clientes. Essas classes são instaladas em um *Servlet Container* ou *Web Container* (Servidor), o que permite à *servlet* tratar as requisições.

1.2 Requisição e resposta

Para promover a interação entre os diversos componentes envolvidos na programação Web dinâmica, a *servlet* deve receber uma requisição e devolver uma resposta ao cliente. Veja um exemplo na Figura “Sistema realizando *Request/Response*”.

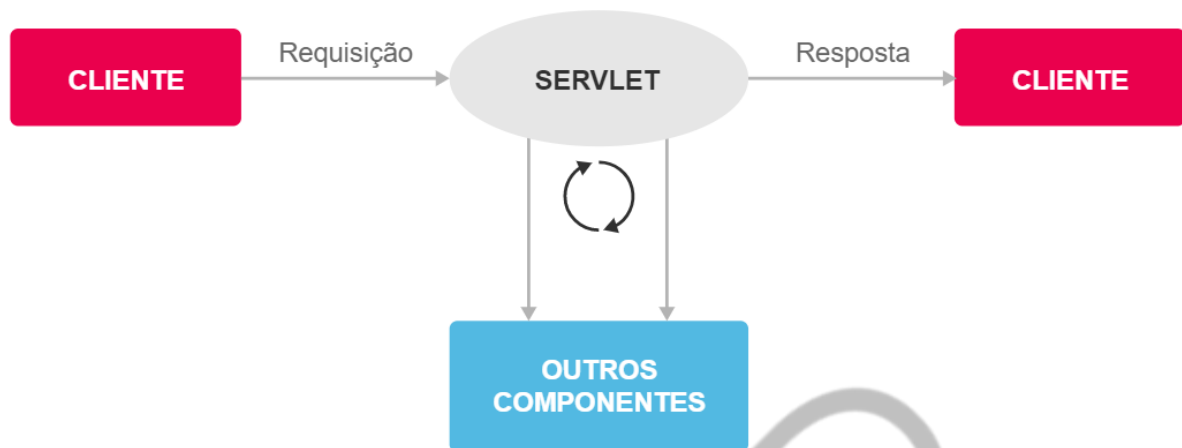


Figura 1 – Sistema realizando *Request/Response*
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

Uma *servlet* tem a função de recuperar as informações enviadas pelo usuário (por meio de uma requisição) e passar essas informações para outros componentes da aplicação para que sejam processadas. Depois, esses componentes devem retornar algum valor para que a *servlet* possa enviar uma resposta para o usuário. Parece ou não parece com o meio de campo de um time de futebol? Poderíamos até ilustrar com uma narração.



Figura 2 – Narração de um jogo de futebol
Fonte: FIAP (2017)

— Olá, iniciamos mais uma transmissão on-line. Aqui quem fala é o seu navegador e vou narrar o processo pelo qual o usuário realiza uma pesquisa.

“E iniciamos mais uma partida em que temos o usuário, que está indeciso sobre o que pesquisar. O sistema está alerta e tenta autocompletar as frases do usuário, mas ele dribla o sistema e escreve a frase por si mesmo. E lá vai ele chegando perto do botão de Enviar, com o mouse ele vai, vai, vai. Clicou, ele clicou!

“CLICOUUUUUUUUU!!!!!!

“Agora vamos ao nosso comentarista. Diz aí, Servidor de Aplicação, o que só você viu.”

— Olha, navegador, eu percebi que o usuário ficou um pouco indeciso na hora de clicar e não aceitou a assistência do autocompletar do sistema. Após o clique, a *Servlet* agarrou a informação, passou para os componentes auxiliares que a processaram, somente depois ela deu a resposta do clique para o usuário, que ficou feliz com a resposta obtida nesse processo de pesquisa realizado aqui hoje e tão brilhantemente narrado por você, navegador.

— Bem amigos, encerramos aqui mais uma transmissão on-line. Quero agradecer a participação do usuário, Servidor de Aplicação, aqui no nosso sistema. Fiquem com Deus e até o próximo processo!

1.3 Requisição a uma *Servlet*

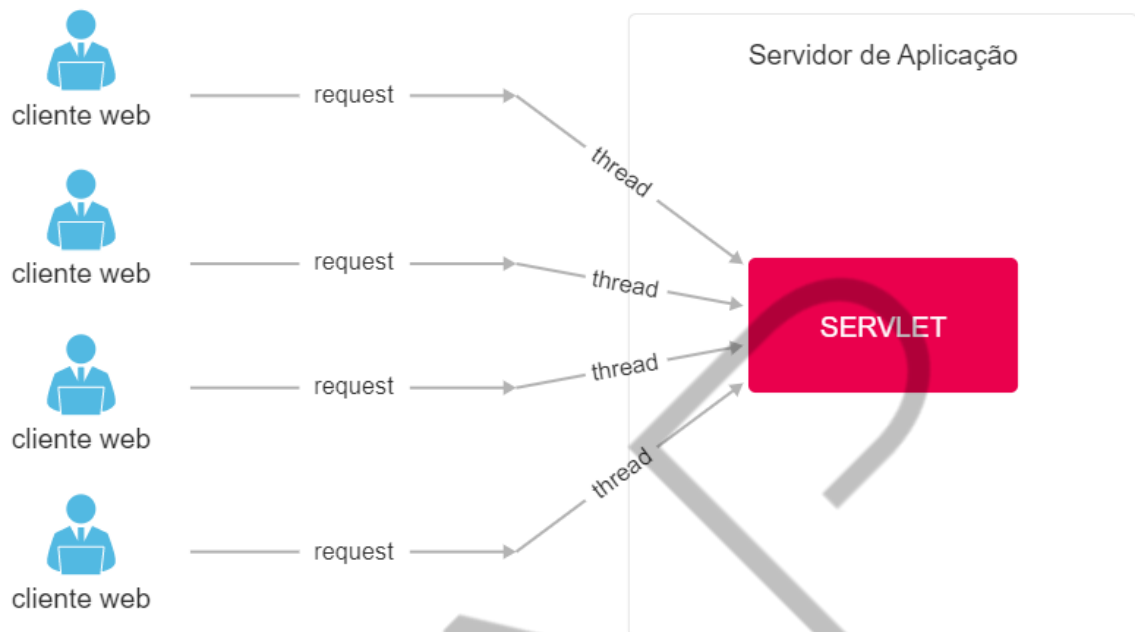


Figura 3 – Exemplo de requisição a uma *servlet*
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

- Cada requisição a uma *servlet* é executada em uma *thread*.
- O objeto *servlet* é único na aplicação.

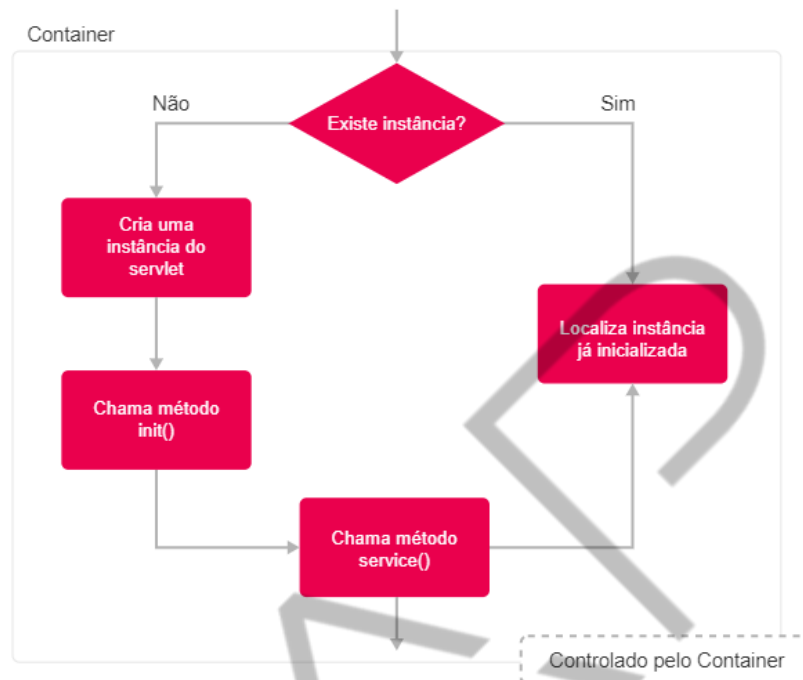
Na Figura “Exemplo de requisição a uma *servlet*”, podemos ver que a *servlet* é acessada por vários clientes simultaneamente, isso é possível pelo fato de terem sido criados *threads* (linhas de execução paralelas) para cada um dos clientes.

1.4 Ciclo de vida de uma *servlet*

Se olharmos à nossa volta, podemos observar diversos seres vivos, cada um com seu próprio ciclo de vida. Igualmente, alguns objetos também têm seu ciclo de utilidade dividido em etapas bem definidas. Logicamente, estou sendo bem pragmático. Isso não é diferente com as nossas *servlets*, veja abaixo:

- As *servlets* são instanciadas pelo *container*, na primeira vez que são acessadas.
- Após iniciadas, as *servlets* podem atender a requisições.

- O *container* decide a hora de destruir as *servlets* (chamando o método `destroy()`).

Figura 4 – Ciclo de vida de uma *servlet*

Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

Para uma classe “se tornar” uma *servlet*, basta ela herdar a classe **HttpServlet**. Essa herança transforma uma simples classe Java em *servlet*, que está pronta para atender às requisições dos clientes. Para isso, a classe deve sobrescrever os métodos **doGet** e/ou **doPost**, dependendo do tipo de requisição que será tratada. O código 2.1 ilustra um exemplo de *servlet* que sobrescreve os métodos **doGet** e **doPost**, assim como o método `init()`, que é chamado pelo *container* após a *servlet* ser instanciada.

```
public class MeuServlet extends HttpServlet {

    public void init(ServletConfig config) throws
ServletException {

    }

    public void destroy() {

    }

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws
ServletException, IOException {

    }

}
```

```
public void doPost(HttpServletRequest request,
    HttpServletResponse
        response) throws ServletException, IOException {
    }
}
```

Código-fonte 1 – Exemplo de uma *Servlet*.
Fonte: Elaborado pelo autor (2016)

ATENÇÃO: No código-fonte “Exemplo de uma *Servlet*”, temos dois métodos dentro da *servlet*.

O `doGet` e o `doPost`, os quais representam métodos do protocolo HTTP, o GET e o POST, pois dependendo de como as informações serão submetidas a partir da página HTML, você terá que esperar essas informações em um desses métodos.

1.5 Sequência de eventos no `HttpServlet`

Podemos ver na Figura “Sequência de eventos em uma *Servlet*” como ocorre a sequência de eventos disparados por meio de um *request* para nossa *servlet*, até atingir um dos dois métodos que vão processar esse *request*.

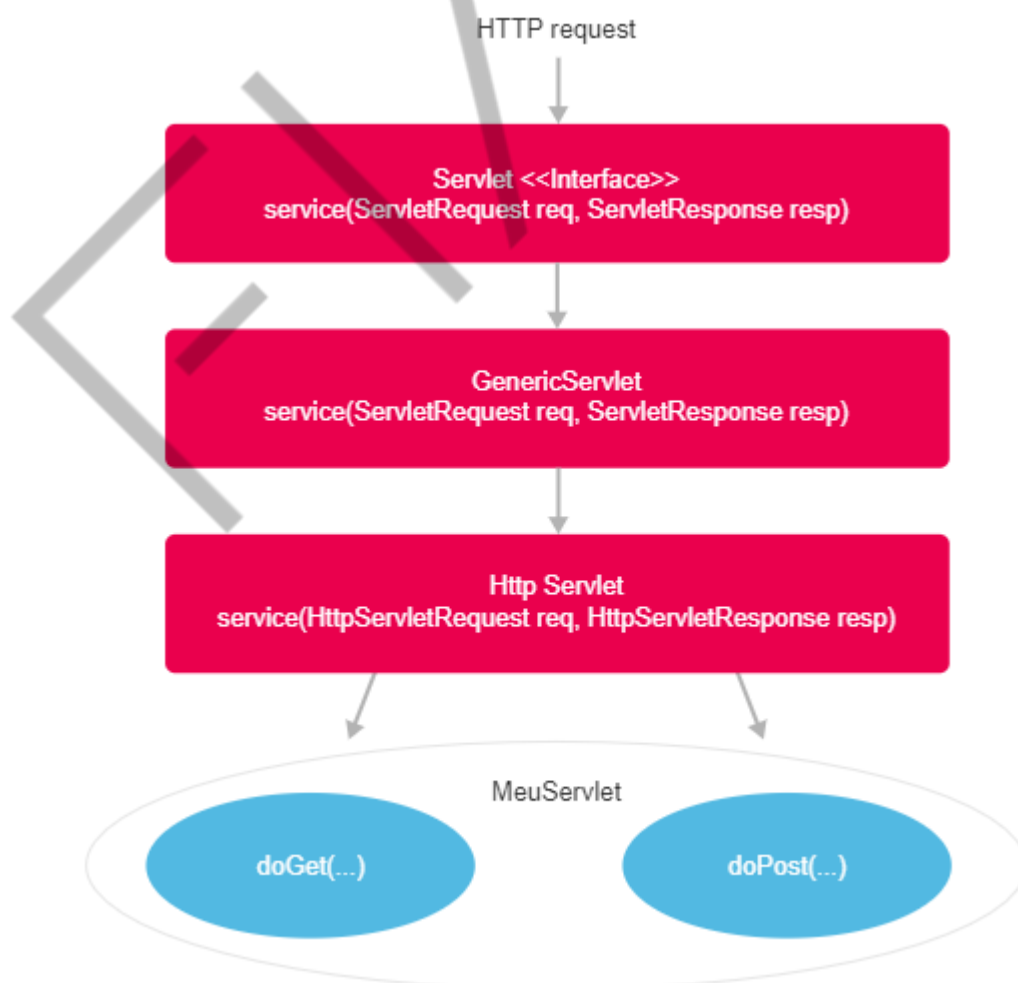


Figura 5 – Sequência de eventos em uma *Servlet*.
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

1.5.1 Interface `HttpServletRequest`

A interface **`HttpServletRequest`** contém vários métodos para lidar com as solicitações do cliente. Sempre que uma *servlet* é invocada, o *container* Web passa os objetos que implementam a interface **`HttpServletRequest`** para o método **`service()`** da *servlet*. A interface `HttpServletRequest`:

- Representa a requisição feita pelo usuário.
- Possibilita obter dados enviados pelo browser, atributos, informações de endereço de IP, protocolo etc.

Possui alguns métodos importantes:

- **Enumeration `getHeaderNames()`** – Obtém o nome de todos os atributos do *header* HTTP passados pelo browser. Com esse método, é possível listar os nomes dos parâmetros que são enviados na requisição. Esses nomes serão utilizados para recuperar os valores dos atributos. Dessa forma, podemos saber o nome do browser do usuário e qual tipo de dado foi enviado, por exemplo.

```
Enumeration<String> h = request.getHeaderNames();  
  
while (h.hasMoreElements()) {  
    System.out.println("Elemento :" + h.nextElement());  
}
```

Código-fonte 2 – Exemplo do método `getHeaderNames()`

Fonte: Elaborado pelo autor (2016)

O resultado da execução do código-fonte “Exemplo do método `getHeaderNames()`” é apresentado abaixo:

```
Saida:  
Elemento :host  
Elemento :user-agent  
Elemento :accept  
Elemento :accept-language
```

```

Elemento :accept-encoding
Elemento :dnt
Elemento :referer
Elemento :cookie
Elemento :connection
Elemento :content-type
Elemento :content-length

```

- **String getHeader(String)** – Obtém o valor de um atributo no header HTTP.

```
System.out.println(request.getHeader("user-agent"));
```

Código-fonte 3 – Exemplo do método getHeader()

Fonte: Elaborado pelo autor (2016)

Observe que é possível recuperar qualquer atributo do cabeçalho HTTP. Os nomes dos atributos foram exibidos na saída da execução do método **getHeaderNames()**. Neste exemplo, obtemos o nome e versão do navegador que realizou a requisição. A seguir, apresentamos um exemplo de saída da execução do código:

```

Saída:
Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101
Firefox/47.0

```

- **getAttribute(String) / setAttribute(String, Object)** – Permite obter e armazenar objetos Java temporariamente na requisição. Esses métodos serão úteis no momento de passar informações da *servlet* para a página web.

```

request.setAttribute("Attr1","1");
request.setAttribute("Attr2","2");
request.setAttribute("Attr3","3");
request.setAttribute("Attr4","4");

System.out.println("Atributo      1      : "      +
request.getAttribute("Attr1"));
System.out.println("Atributo      2      : "      +
request.getAttribute("Attr2"));

```

```
System.out.println("Atributo      3      : "      +
request.getAttribute("Attr3"));
System.out.println("Atributo      4      : "      +
request.getAttribute("Attr4"));
```

Código-fonte 4 – Exemplo do método `setAttribute()` e `getAttribute()`

Fonte: Elaborado pelo autor (2016)

Após armazenar os valores na *request* com o método **`setAttribute()`**, o código utiliza o método **`getAttribute()`** para recuperar os valores armazenados e exibe-os no console do eclipse. O resultado da execução pode ser visto a seguir:

```
Saída:
Atributo 1 :1
Atributo 2 :2
Atributo 3 :3
Atributo 4 :4
```

- **Enumeration `getAttributeNames()`** – Obtém todos os nomes dos atributos armazenados na requisição.

```
Enumeration<String> atrib = request.getAttributeNames();

while (atrib.hasMoreElements()) {
    System.out.println("Elemento      : "      +
atrib.nextElement());
}
```

Código-fonte 5 – Exemplo do método `getAttributeNames()`

Fonte: Elaborado pelo autor (2016)

O código-fonte “Exemplo do método `getAttributeNames()`” recupera todos os nomes de atributos armazenados no *request* e imprime no console. Resultado da execução:

```
Saída:
Elemento :Attr1
Elemento :Attr2
Elemento :Attr3
Elemento :Attr4
```

- **Cookie[] getCookies()** – Obtém os *cookies* enviados pelo browser. *Cookies* são informações que o browser do usuário pode armazenar por um período maior de tempo, muito útil para identificar o usuário e suas preferências. Dois exemplos bem comuns para o uso de *cookie* são: armazenamento de credencias de acesso que são recuperados a cada acesso ao site e para guardar algumas preferências de buscas de produtos nos mecanismos de busca.

```
Cookie[] cookie = request.getCookies();  
for(Cookie c : cookie){  
    System.out.println("Nome do Cookie :" + c.getName());  
    System.out.println("Nome do Cookie :" + c.getValue());  
}
```

Código-fonte 6 – Exemplo do método `getCookies()`
Fonte: Elaborado pelo autor (2016)

- **HttpSession getSession()** – Obtém a sessão do usuário. Esse objeto pode armazenar informações do usuário por um período maior de tempo na aplicação Java. A sessão é um recurso muito comum para armazenar itens de um carrinho de compra em um portal de venda de produtos.

```
HttpSession sessao = request.getSession();
```

Código-fonte 7 – Exemplo do método `getSession()`
Fonte: Elaborado pelo autor (2016)

- **String getParameter(String)** – Obtém informações submetidas de um formulário ou link HTML (abordaremos esse método com mais detalhes no decorrer do capítulo).

```
String nome = request.getParameter("nome");
```

Código-fonte 8 – Exemplo do método `getParameter()`
Fonte: Elaborado pelo autor (2016)

Esse método é o principal na relação entre as páginas HTML/JSP e as servlets, pois é por meio dele que recuperamos os valores de formulários, por exemplo.

A Figura “Exemplo de uma URL para requisição” apresenta as partes de uma URL utilizada para realizar uma requisição no servidor.

Dado o request HTTP:

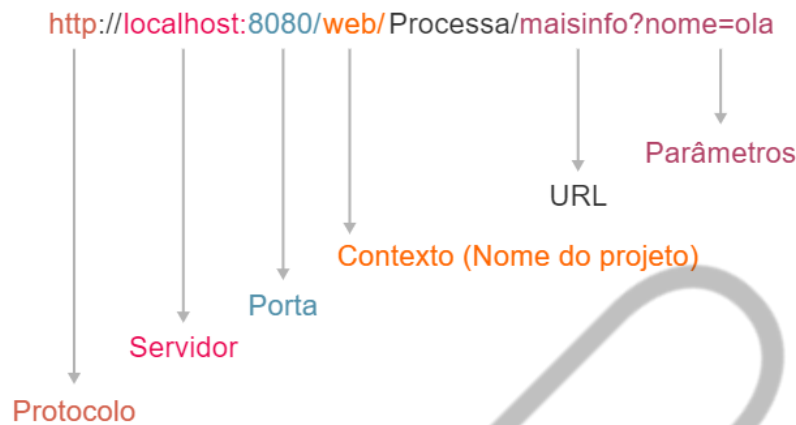


Figura 6 – Exemplo de uma URL para requisição
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

Já a Figura “Métodos para recuperar cada informação da requisição” detalha os métodos da interface **HttpServletRequest** que são utilizados para recuperar cada informação da requisição.

Obter dados do request com os seguintes métodos:

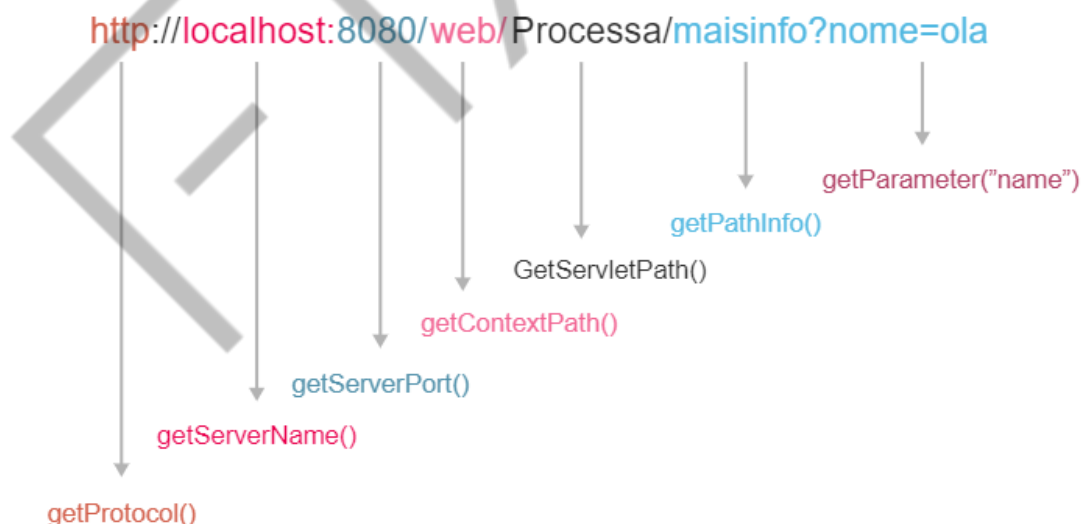


Figura 7 – Métodos para recuperar cada informação da requisição
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

1.5.2 Interface HttpServletResponse

A interface **HttpServletResponse** contém vários métodos que permitem à *servlet* responder às solicitações do cliente, representa a resposta que será enviada ao cliente. Possui alguns métodos:

- **addCookie(Cookie)** – Adiciona um *cookie* na resposta que será enviado para o browser.

```
Cookie cookie = new Cookie("cookieTeste1", "Alexandre|Carlos");  
response.addCookie(cookie);
```

Código-fonte 9 – Exemplo de método addCookie()

Fonte: Elaborado pelo autor (2016)

- **setContentType(String)** – Informa o tipo de retorno que será enviado para o browser. Podemos determinar o tipo como uma página HTML, pdf, texto, imagem, json etc.

```
response.setContentType("text/html");
```

Código-fonte 10 – Exemplo de método setContentType()

Fonte: Elaborado pelo autor (2016)

- **getWriter()** – Obtém o objeto **PrintWriter** para envio de texto/html para o browser. Com ele é possível escrever tags HTML e textos que serão exibidos para o usuário.

```
response.setContentType("text/html");  
PrintWriter out = response.getWriter();  
out.println("<html>");  
out.println("<body>");  
out.println("Exemplo Servlet");  
out.println("</body>");  
out.println("</html>");
```

Código-fonte 11 – Exemplo de método getWriter()

Fonte: Elaborado pelo autor (2016)

2 APLICAÇÃO WEB

Depois que falamos um pouco sobre as *servlets* e os principais métodos das interfaces que representam o *request* e o *response* (`HttpServletRequest` e `HttpServletResponse` respectivamente), vamos implementar uma *servlet* que interage com uma página HTML.

2.1 Recuperação de parâmetros

Imagine que você criou uma página em HTML com diversos elementos/*tags*. Sabemos que cada um deles exerce algum tipo de função ao ser interpretado pelo navegador e que todos possuem atributos, e são eles que possibilitam a captura das informações no momento da requisição (*request*) em nossas *servlets*.

Na Figura “Exemplo de formulário HTML” é apresentado um formulário para validação de *login* e senha do usuário. O código-fonte “Exemplo de implementação do formulário HTML” exibe a implementação em HTML desse formulário.


A imagem mostra um formulário web com o título "Validação" no topo. Abaixo do título, há o rótulo "Login" seguido por um campo de entrada de texto. Logo abaixo, há uma linha horizontal separadora. Segue o rótulo "Senha" e outro campo de entrada de texto. Na base do formulário, há um botão com o texto "Enviar".

Figura 8 – Exemplo de formulário HTML
Fonte: Elaborado pelo autor (2016)

```
<form action="ExemploServlet" method="post">
  <fieldset>
    <legend>Validação</legend>
    <label>Login</label>
    <input type="text" name="login">
    <hr />
    <label>Senha</label>
```

```
<input type="text" name="senha">
</fieldset>
<input type="submit" value="Enviar">
</form>
```

Código-fonte 12 – Exemplo de implementação do formulário HTML
Fonte: Elaborado pelo autor (2016)

Note que, no código-fonte “Exemplo de implementação do formulário HTML”, o elemento **form** utiliza o método **POST** do **HTTP** (propriedade method), pois este é o termo mais adequado para submeter / enviar um formulário HTML. O método **GET** é mais utilizado para recuperar informações – uma busca, por exemplo.

Vale observar também a propriedade **action** do formulário, essa configuração define o destino das informações do formulário, ou seja, a URL que será utilizada para enviar os dados para o servidor, no código-fonte acima, o **ExemploServlet** representa o endereço ou o mapeamento de uma *servlet* no sistema.

2.2 Mapeamento das *Servlets*

No formulário que está sendo apresentado no código-fonte “Propriedade action da tag <form> do HTML”, podemos ver que a propriedade **action** aponta para o nome da *servlet*. É isso mesmo?

```
<form action="ExemploServlet" method="post">
  <!--Código...-->
</form>
```

Código-fonte 13 – Propriedade action da tag <form> do HTML
Fonte: Elaborado pelo autor (2016)

É, mais ou menos, mais ou menos!

Vamos lá! Sempre que você criar uma *servlet*, a IDE não tem como saber qual será o endereço, mapeamento ou apelido que você dará à sua *servlet*, então, a IDE assume que o mapeamento será o próprio nome da classe.

O mapeamento pode ser realizado por meio do arquivo **web.xml** da aplicação ou de **annotations**. Anotações em Java começam com o caractere @ (arroba). Elas têm a função de colocar mais informações em uma classe, método, atributo, parâmetro de método etc. Nesse caso, a anotação **@WebServlet** está informando à **url** que a classe *servlet* está esperando para ser invocada.

```
@WebServlet("/ExemploServlet")
public class ExemploServlet extends HttpServlet {
    //.....
}
```

Código-fonte 14 – Exemplo de uma *servlet* com seu mapeamento inicial
Fonte: Elaborado pelo autor (2016)

Porém, deixar o mesmo nome da classe na url não é bom e podemos ver de longe o porquê. Primeiro, você vai ter um formulário html a que todo mundo vai ter acesso, e quem vai estar lá? O nome da sua *servlet* exposto para o mundo! Eu e 99,99% da comunidade javanesa de tecnologia para a web achamos que isso pode ser um comportamento de risco para o seu sistema, então seria de bom-tom mudar esse mapeamento para algo mais amigável e totalmente diferente do nome da *servlet*, o que acha? Vamos mudar para “/exemplo” – não se esqueça da barra, pois isso indica ao servidor que esse é um mapeamento da *servlet*.

```
@WebServlet("/exemplo")
public class ExemploServlet extends HttpServlet {
    //.....
}
```

Código-fonte 15 – Exemplo de uma *servlet* com seu mapeamento alterado
Fonte: Elaborado pelo autor (2016)

Pronto, estamos jogando com o regulamento debaixo do braço. Agora, nossa *servlet* será conhecida em nossa aplicação como “/exemplo”. Dessa forma, basta colocar esse nome nos links e formulários para submeter as informações.

```
<form action="exemplo" method="post">
    <!--Código.-->
</form>
```

Código-fonte 16 – Propriedade action com o valor ajustado para o mapeamento da *servlet*
Fonte: Elaborado pelo autor (2016)

No código-fonte abaixo é apresentado o formulário HTML com o valor da propriedade **action** ajustado para o mapeamento da nossa *servlet*. Este é o momento de implementar a *servlet* que receberá a requisição desse formulário.

```
<form action="exemplo" method="post">
  <fieldset>
    <legend>Validação</legend>
    <label>Login</label>
    <input type="text" name="login">
    <hr />
    <label>Senha</label>
    <input type="text" name="senha">
  </fieldset>
  <input type="submit" value="Enviar">
</form>
```

Código-fonte 17 – Exemplo de formulário HTML para recuperação de parâmetros
Fonte: Elaborado pelo autor (2016)

O código-fonte “Exemplo de Servlet para recuperação de parâmetros” confirma o que nós já sabíamos: que a nossa *servlet* joga muito! Quer dizer, controla o fluxo das informações em nossa aplicação web. Podemos ver claramente os parâmetros que foram passados por meio do *request* via método **POST**, basta olhar para os parâmetros que os métodos **`request.getParameter("login")`** e **`request.getParameter("senha")`** estão recebendo, olhe para o código HTML mais uma vez e veja o nome dos campos do formulário, eles estão representados aqui como parâmetros dos métodos, já com suas informações para serem coletadas, armazenadas e processadas.

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    String login = request.getParameter("login");
    String senha = request.getParameter("senha");

    if(login.equals("userx") && senha.equals("xxxxx")){
        response.sendRedirect("sucesso.html");
    }else{
        response.sendRedirect("acesso_negado.html");
    }
}
```

Código-fonte 18 – Exemplo de Servlet para recuperação de parâmetros
Fonte: Elaborado pelo autor (2016)

No código-fonte “Exemplo de Servlet para recuperação de parâmetros”, estamos ainda realizando uma pequena validação para verificar se a senha e o nome do usuário estão corretos.

Para devolver uma resposta para o usuário, vamos redirecioná-lo para páginas HTML estáticas, ou seja, páginas que já existem e não foram geradas naquele momento. Logo mais, vamos trabalhar com páginas web dinâmicas.

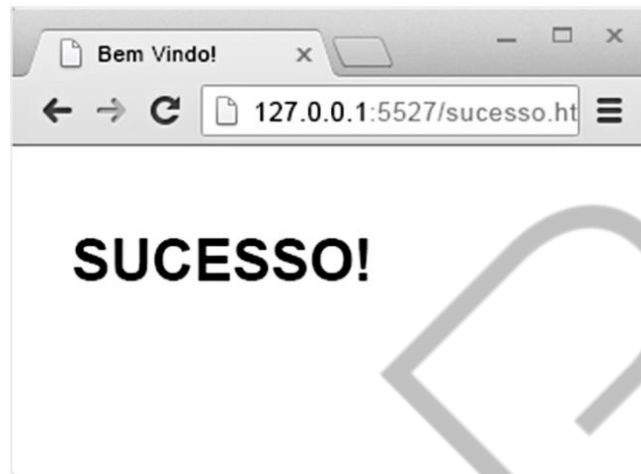


Figura 9 – Página de resposta de sucesso
Fonte: Elaborado pelo autor (2016)

A Figura “Página de resposta de sucesso” mostra uma página HTML estática que existe em nosso sistema e para onde o cliente foi direcionado após ter sido validado com sucesso. Caso não passe na validação, o cliente será direcionado para a página de acesso negado, que também já existe em nosso sistema.



Acesso Negado!

Figura 10 – Página de resposta de erro
Fonte: Elaborado pelo autor (2016)

O que vimos foi o processo de **request** e **response**, em que o cliente gera uma requisição para a nossa *servlet* e ela recebe a requisição originada em um dos métodos do HTTP (**GET** ou **POST**). Na *servlet* existem dois métodos – **doGet** e

doPost – responsáveis por receber os parâmetros que vêm no **request** para a *servlet* e geram o **response** para o cliente, que sempre será HTML.

A figura abaixo apresenta uma visão geral do processamento do *request* pela *servlet*, mostra o *request* que foi gerado pelo formulário e enviado para a *servlet*, e depois, exibe a resposta para o usuário, redirecionando-o para uma página HTML.

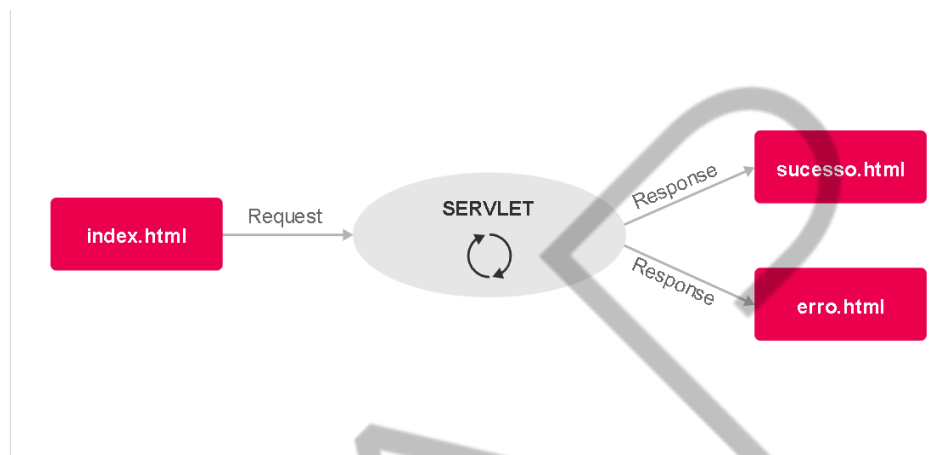


Figura 11 – Visão geral do processamento da *request* pela *servlet*
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

2.3 Servlet request

Acabamos de ver a passagem de parâmetros (valores) de páginas HTML para a nossa *servlet* e a geração de um *response* para o cliente com páginas HTML estáticas, que já estavam preparadas para esse fim.

Mas, e se você precisar passar uma informação dinamicamente? Por exemplo, em uma página, você tem uma calculadora simples para a realização de operações matemáticas e quer saber o resultado das operações que você fizer nessa calculadora. A resposta para o cliente não poderia ser previamente criada, porque os resultados são infinitos, então para esse problema, vamos utilizar nossa *servlet* como um *requester* ou cliente.

Sim, foi isso mesmo que você leu, a partir de agora, a *servlet* enviará *requests* para outras páginas, passando um determinado valor, como o resultado de uma operação matemática.

Vamos então demonstrar isso com uma calculadora simples em HTML, observe o código-fonte “Exemplo de formulário HTML (calculadora) para a passagem de parâmetros”.

```
<form action="exemplo" method="post">
  <fieldset>
    <legend>CALCULAR</legend>
    <label>Valor - 1</label>
    <input type="text" name="valor1">
    <hr />
    <label>Operação</label>
    <select name="operacao">
      <option value="soma">+</option>
      <option value="subtracao">-</option>
      <option value="multiplicacao">*</option>
      <option value="divisao">/</option>
    </select>
    <hr />
    <label>Valor - 2</label>
    <input type="text" name="valor2">
  </fieldset>
  <input type="submit" value="Enviar">
</form>
```

Código-fonte 19 – Exemplo de formulário HTML (calculadora) para a passagem de parâmetros
Fonte: Elaborado pelo autor (2016)

A Figura “Resultado da execução do código do formulário HTML” exibe a página HTML com os campos para o usuário efetuar a digitação e selecionar a operação desejada.

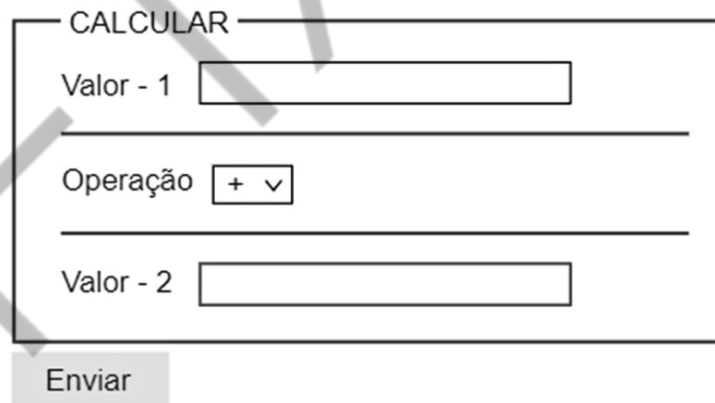


Figura 12 – Resultado da execução do código do formulário HTML
Fonte: Elaborado pelo autor (2016)

O processo de passagem de parâmetros é o mesmo, nada muda, porém, adicionaremos agora algumas informações na *servlet*. Vamos dar uma olhada em nossa *servlet* com esse novo processo.

```
public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {

    int result = 0;

    if(request.getParameter("operacao").equals("soma"))
    {

        result =
Integer.parseInt(request.getParameter("valor1")) +
Integer.parseInt(request.getParameter("valor2"));

        }else
if(request.getParameter("operacao").equals("subtracao")){

        result =
Integer.parseInt(request.getParameter("valor1")) -
Integer.parseInt(request.getParameter("valor2"));

        }else
if(request.getParameter("operacao").equals("multiplicacao")){

        result =
Integer.parseInt(request.getParameter("valor1")) *
Integer.parseInt(request.getParameter("valor2"));

        }else
if(request.getParameter("operacao").equals("divisao")){

        result =
Integer.parseInt(request.getParameter("valor1")) /
Integer.parseInt(request.getParameter("valor2"));
        }

        request.setAttribute("resultado", result);

        request.getRequestDispatcher("resposta.jsp").forward(request, response);
    }
}
```

Código-fonte 20 – Exemplo do método doPost da *servlet* recebendo os parâmetros do formulário
Fonte: Elaborado pelo autor (2016)

No código-fonte “Exemplo do método doPost da *servlet* recebendo os parâmetros do formulário”, a *servlet* recebe os parâmetros que são os dois números e a operação matemática que será realizada. O primeiro a ser avaliado é o parâmetro operação, para determinar qual operação será executada com os demais valores passados.

Por fim, foi criado um atributo no *request*. O atributo é criado no estilo chave e valor, **request.setAttribute("resultado", result)**, “resultado” é a chave e o *result* o valor. Esse valor foi adicionado no *request* que será encaminhado para a página **resposta.jsp** por meio do seguinte método:

```
request.getRequestDispatcher("resposta.jsp").forward(request, response).
```

O *forward* é um método encapsulado que leva as informações para a página de destino, no nosso caso, a página **resposta.jsp**. Agora, falta recuperar as informações na página de destino. Como vamos realizar esse processo?

Simple, antes de tudo, preciso dizer que somente páginas JSP (Java Server Pages) têm a capacidade de recuperar os dados enviados pela *servlet*. Os arquivos HTML não possuem tal característica. Por ora, saiba que as páginas JSP trabalham em conjunto com as *servlets*. Logo, teremos um capítulo totalmente dedicado a esse assunto.

Além de recuperar as informações em uma página JSP, precisamos utilizar uma linguagem nativa desse tipo de página, chamada EL (*Expression Language*), cujo símbolo de utilização é: **\${ }**. Agora que já sabemos o que utilizar, vamos ao exemplo da página **resposta.jsp**.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
  <title>Resposta</title>
</head>
<body>

  <h1>Resultado da Operação</h1>
  <p> ${ resultado } </p>

</body>
</html>
```

Código-fonte 21 – Exemplo da página resposta.jsp com a EL
Fonte: Elaborado pelo autor (2016)

O código-fonte “Exemplo da página resposta.jsp com a EL” ilustra a página resposta JSP com a EL (*Expression Language*) recuperando o valor do atributo resultado, que foi criado na *servlet* e enviado no *request*.

Agora, o resultado final desse processo é apresentado nas figuras “Exemplo do formulário HTML (calculadora) enviando as informações” e “Página resposta.jsp após ter recebido o atributo enviado pela *servlet*”.

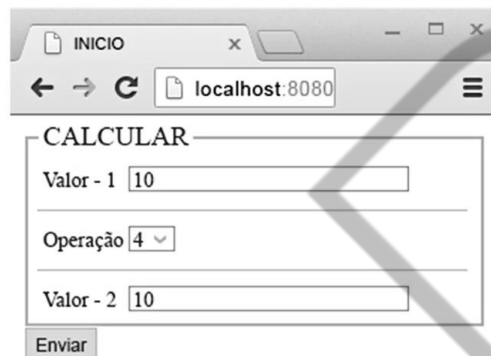
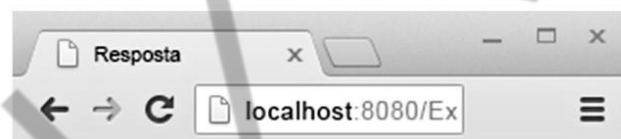


Figura 13 – Exemplo do formulário HTML (calculadora) enviando as informações
Fonte: Elaborado pelo autor (2016)



Resultado da Operação

20

Figura 14 – Página resposta.jsp após ter recebido o atributo enviado pela *servlet*
Fonte: Elaborado pelo autor (2016)

A visão geral do processo pode ser visualizada na Figura “Visão geral do fluxo de processamento da *request*”:

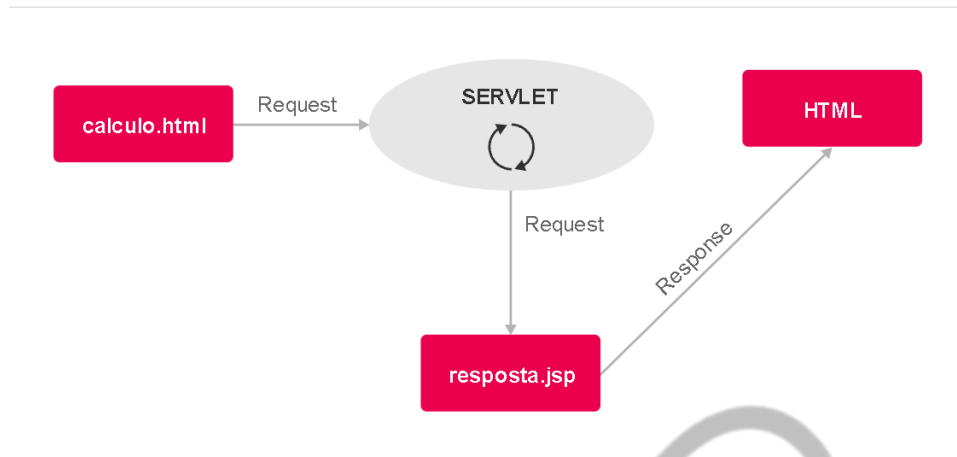


Figura 15 – Visão geral do fluxo de processamento da *request*
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

A página ***calculo.html*** envia uma requisição assim que o usuário clica no botão calcular do formulário. Após o envio do *request*, a *servlet* recebe as informações e realiza o cálculo necessário. Por fim, a *servlet* adiciona na requisição o valor da resposta do cálculo e encaminha a requisição para a página JSP. A página JSP é processada, recupera a informação adicionada pela *servlet* (atributo) e gera o HTML final, que é enviado no *response* para o usuário.

Lembre-se de que aquele HTML gerado como *response* é HTML puro, que o usuário vê ao final de todo o processo, pois ele sempre vai ter um *response*, então considere a Figura “Página resposta.jsp após ter recebido o atributo enviado pela *servlet*” como o equivalente daquele HTML.

2.4 Request através de links

Até o momento, trabalhamos com formulários para enviar as requisições, porém, existe outra maneira de realizarmos *requests* para nossas *servlets*. Sim, o outro meio de realizar requisição é por meio de links, pois, quando digitamos um endereço no browser estamos realizando uma requisição, o endereço que aponta para uma página envia uma requisição para o servidor, pedindo que retorne o conteúdo da página no nosso navegador. Um link nada mais é do que um endereço que será utilizado para realizar uma requisição no servidor.

Dessa forma, podemos apontar o link para uma página ou para a nossa *servlet*, mas aí vocês podem perguntar: “Como? Mostra!”. Está bem, vou mostrar no código-fonte “Exemplo de link apontando para nossa *servlet*”.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Resposta</title>
</head>
<body>

    <p> <a href="exemplo">LINK PARA A SERVLET</a></p>

</body>
</html>
```

Código-fonte 22 – Exemplo de link apontando para nossa *servlet*
Fonte: Elaborado pelo autor (2016)

Viu? Fácil, não é?!

Vocês podem se perguntar como podemos enviar parâmetros para as *servlets*, igual fizemos com os formulários. Nos links, podemos passar valores para as *servlets*, utilizando o próprio endereço (URL), da mesma forma que uma requisição com um formulário configurado com o método GET. Vamos dar uma olhada aqui no código-fonte “Exemplo de link apontando para nossa *servlet* e passando um parâmetro”.

```
<p>    <a href="exemplo?parametro=valor">LINK    PARA    A
SERVLET</a></p>
```

Código-fonte 23 – Exemplo de link apontando para nossa *servlet* e passando um parâmetro
Fonte: Elaborado pelo autor (2016)

Pronto, bastou colocar um sinal de interrogação (?) depois do nome da *servlet* para podermos adicionar os parâmetros. Um parâmetro é formado pelo nome seguido do sinal de igual e seu valor, conforme o exemplo: *servlet?parametro=valor*.

Entretanto, e se precisar passar mais de um parâmetro, como posso fazer? Simples, basta colocar um E comercial (&) depois do valor do último parâmetro para começar um novo:

```
exemplo?parametro1=valor&parametro2=valor
```

Nunca se esqueça do sinal de interrogação (?) para separar os parâmetros da *servlet* e o E comercial (&) para separar os parâmetros.

ATENÇÃO: Os links, por padrão, são submetidos por meio do protocolo HTTP, pelo método GET, então você sempre deve receptionar os links no método doGet de nossa *servlet*.

Vamos a mais um exemplo:

```
<p>      <a      href="exemplo?nome=Alexandre">LINK      PARA      A  
SERVLET</a></p>
```

Código-fonte 24 – Exemplo de link apontando para nossa *servlet* e passando um parâmetro
Fonte: Elaborado pelo autor (2016)

No código-fonte “Exemplo de link apontando para nossa *servlet* e passando um parâmetro”, criamos um link em uma página HTML para enviar um parâmetro para nossa *servlet*. Já na Figura Resultado da execução do código do formulário HTML, apresentamos o resultado do código HTML exibido no browser.

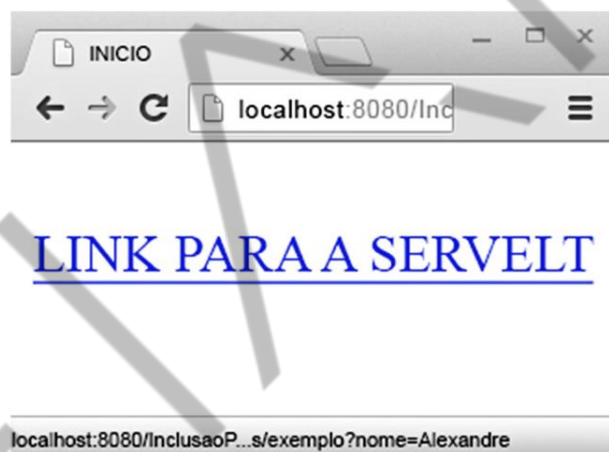


Figura 16 – Link apontando para a *servlet* com o parâmetro
Fonte: Elaborado pelo autor (2016)

Quando o *request* chega à *servlet*, realizamos o mesmo procedimento que já conhecemos, a única coisa que mudou foi que passamos a utilizar o método **doGet** em vez do **doPost** da *servlet*. Dessa forma, podemos observar no código-fonte 2.25 que estamos recebendo o parâmetro “**nome**” na *servlet* e adicionado um atributo “**conteudoLink**” no *request* para que a página JSP possa exibi-lo. Por fim, é feito o redirecionamento do *request* para a página JSP, assim a página recupera o atributo enviado pelo *servlet* e gera o HTML final, que será exibido para o usuário.

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    //Parâmetro enviado através do request por um link
    String parametro = request.getParameter("nome");

    //Criando um atributo no request para encaminhar para
    a página de resposta
    request.setAttribute("conteudoLink", parametro);

    //Realizando o redirecionamento para a página
    resposta.jsp e
    // encaminhando as informações no request junto com
    o response.

    request.getRequestDispatcher("resposta.jsp").forward(request,
    response);
}
```

Código-fonte 25 – Exemplo: código na *servlet* recebendo/repassando o parâmetro enviado por link
Fonte: Elaborado pelo autor (2016)

Podemos observar, no exemplo do código-fonte “Código da página resposta.jsp recuperando atributo por EL passada pela *servlet*”, a página JSP recuperando a informação enviada por um atributo pela *servlet*. Esse conteúdo é colocado em um parágrafo na página HTML.

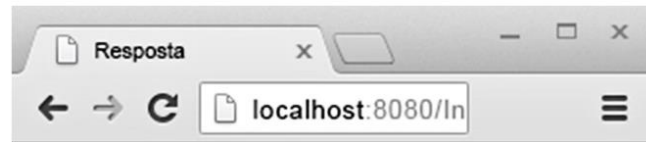
```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Resposta</title>
</head>
<body>

    <h1>Conteúdo do Link</h1>
    <p> ${ conteudoLink }</p>

</body>
</html>
```

Código-fonte 26 – Código da página resposta.jsp recuperando atributo por EL passada pela *servlet*
Fonte: Elaborado pelo autor (2016)

E, por fim, apresentamos, na Figura “Página resposta.jsp apresentando o resultado em HTML ao cliente”, o HTML final que será exibido para o usuário.



Conteúdo do Link

Alexandre

Figura 17 – Página resposta.jsp apresentando o resultado em HTML ao cliente
Fonte: Elaborado pelo autor (2016)

Para ilustrar o processo acima, vamos analisar a Figura “Diagrama ilustrando o *request* gerado através de um link”.

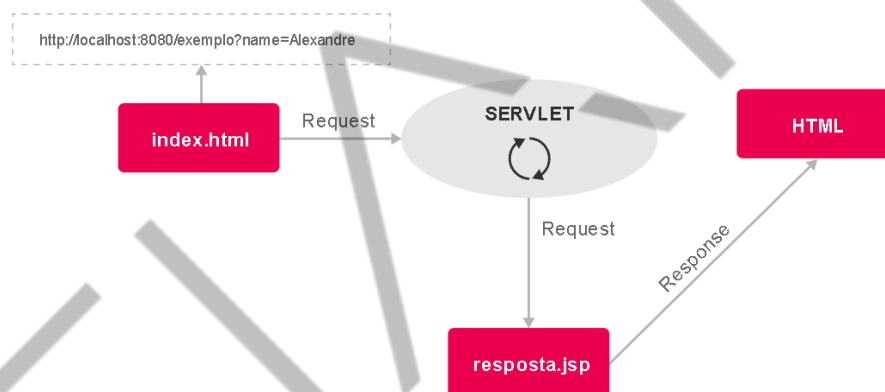


Figura 18 – Diagrama ilustrando o *request* gerado através de um link
Fonte: Elaborado pelo autor (2016), adaptado por FIAP (2017)

Nessa figura, fica claro que praticamente não existem diferenças entre *requests* gerados entre links e formulários, pois eles são processados da mesma maneira, as diferenças são no método HTTP que será utilizado e na quantidade de caracteres suportados, o verbo GET permite até 2048 caracteres na URL e uma requisição feita no verbo POST não possui limitação. Os links são sempre no método **doGet** e o formulário pode ser qualquer um dos dois (doGet ou doPost), vai depender da configuração do formulário no atributo **method**. Esse recurso é muito útil. Desde que você utilize a lógica certa, as possibilidades são infinitas.

Agora que já visualizamos esse novo mundo das aplicações Java Web e alguns de seus componentes, tenho a certeza de que você já deve estar pensando onde

aplicar tudo isso. Sabemos agora que a *servlet* é a base de tudo, se você entendeu essa tecnologia, as demais que dependem dela vão com certeza ficar mais fáceis.

2.5 Prática!

Vamos desenvolver um exemplo para praticar, porque é praticando que se aprende. Primeiramente, vamos criar um projeto no eclipse. O projeto será do tipo “Dynamic Web Project”. Comece escolhendo a opção File > New > Dynamic Web Projeto, conforme a Figura “Criando um projeto web no eclipse – Parte 1”.

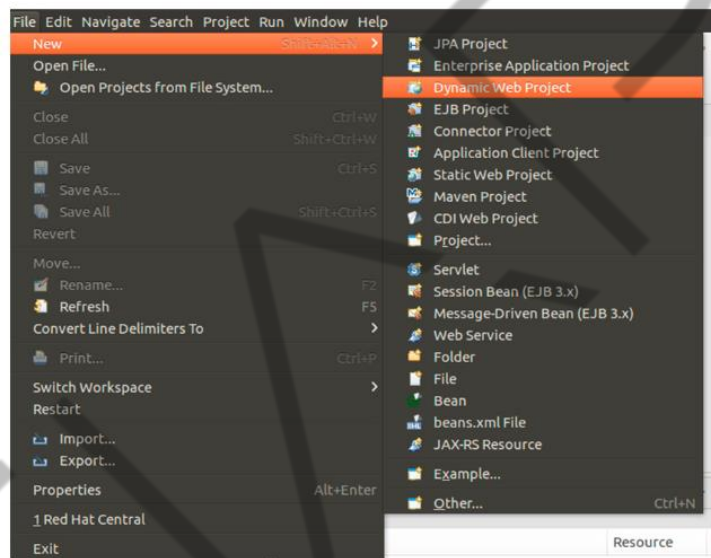


Figura 19 – Criando um projeto web no eclipse – Parte 1
Fonte: Elaborado pelo autor (2017)

Agora é o momento de atribuir um nome ao projeto e configurar o servidor. Para isso, clique no botão “New Runtime...” (Figura “Criando um projeto web no eclipse – Parte 2”).

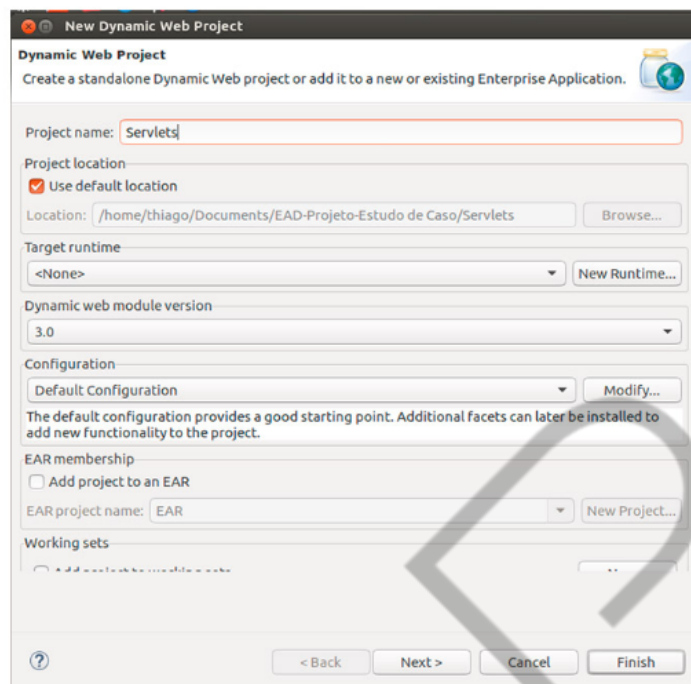


Figura 20 – Criando um projeto web no eclipse – Parte 2
Fonte: Elaborado pelo autor (2017)

Conforme já falamos, vamos utilizar o Apache Tomcat. Escolha a versão do servidor para que você faça o download posteriormente e clique em next (Figura “Configurando o servido no projeto – Parte 1”).

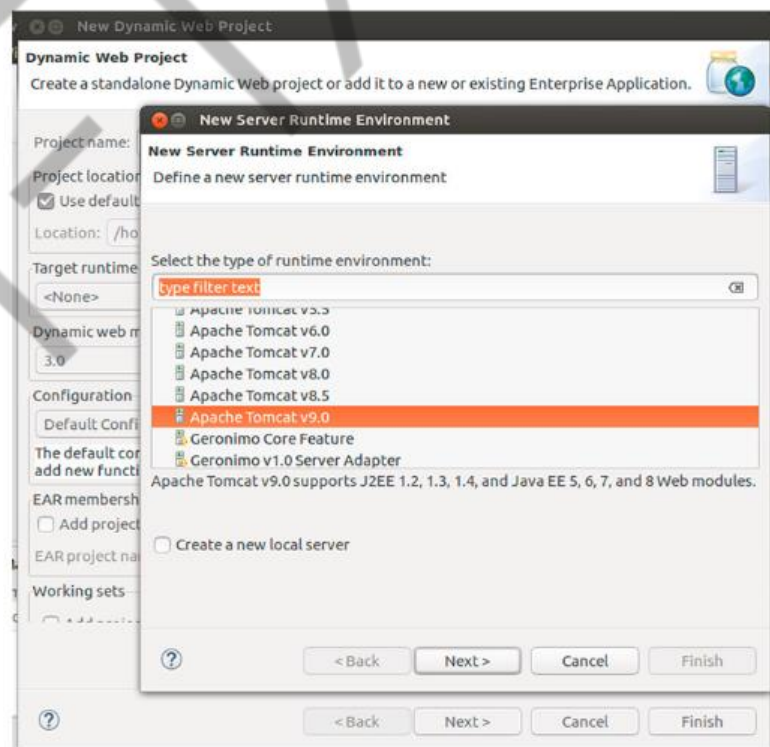


Figura 21 – Configurando o servidor no projeto – Parte 1
Fonte: Elaborado pelo autor (2017)

Neste momento, precisamos informar onde está o servidor. Provavelmente, você realizou o download do Tomcat e descompactou o arquivo. É esse o diretório que você precisa utilizar para configurar o servidor. Para isso, clique no botão *browse* e navegue até o diretório do Tomcat (Figura “Configurando o servidor no projeto – Parte 2”).

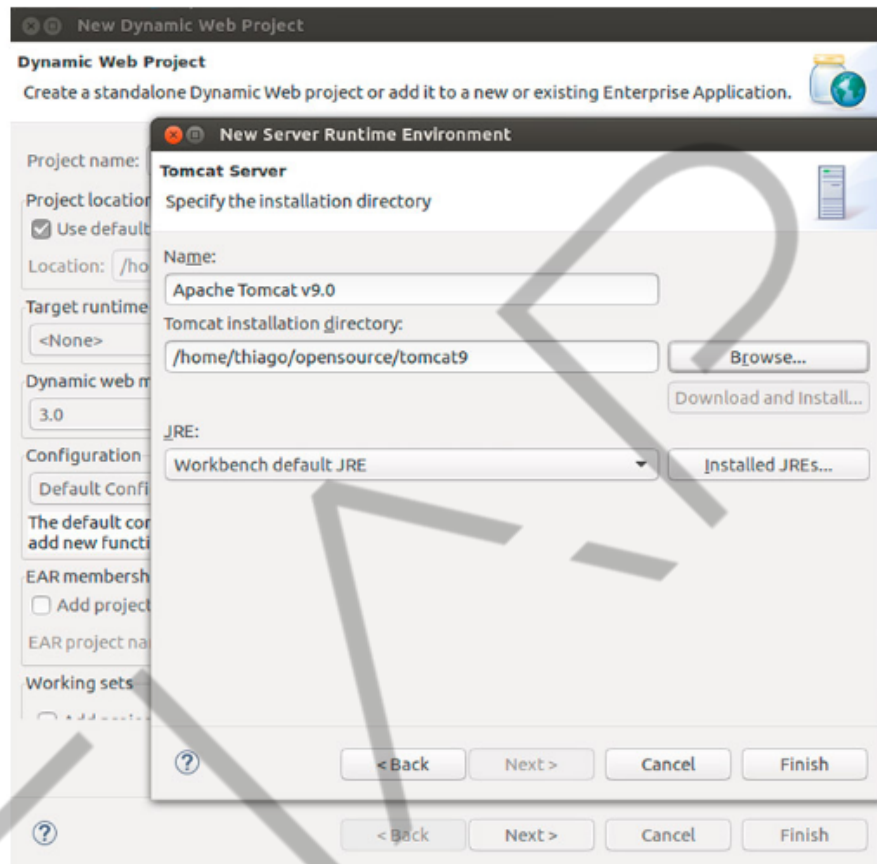


Figura 22 – Configurando o servidor no projeto – Parte 2
Fonte: Elaborado pelo autor (2017)

Finalizada a criação do servidor, podemos visualizar a configuração do Tomcat na área de *Target Runtime* (Figura “Visão geral da primeira configuração do projeto web”). Após essas configurações que acabamos de fazer, podemos então ir para o próximo passo. Clique no botão *next*, nessa tela, temos a opção de configurar a pasta na qual estarão os códigos-fontes e onde ficarão os arquivos compilados, nesse momento não vamos alterar nada, clique novamente em *next*.

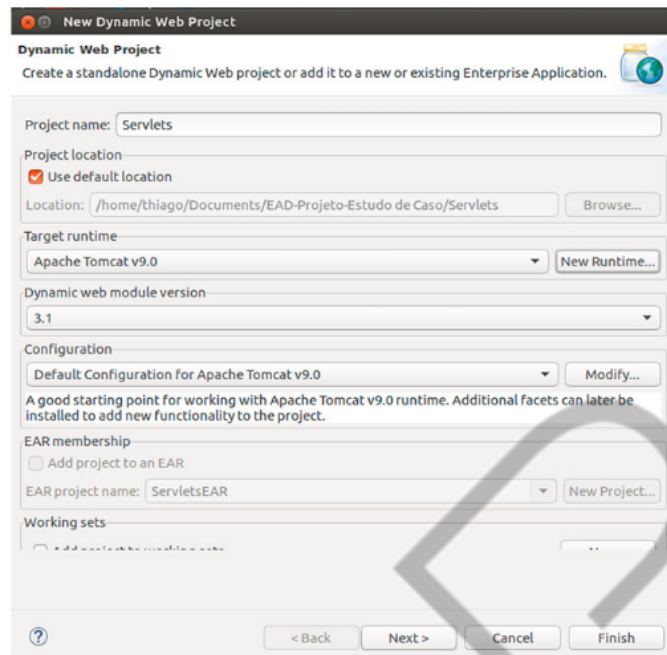


Figura 23 – Visão geral da primeira configuração do projeto web
Fonte: Elaborado pelo autor (2017)

Nessa última tela, configuramos o contexto e o diretório em que ficarão os arquivos web. Porém a configuração padrão já é suficiente, dessa forma, não precisamos alterá-las. Ainda nessa tela, marque a opção “Generate web.xml deployment descriptor” (Figura “Criando um projeto web no eclipse – Parte 3”). Essa opção cria o arquivo **web.xml**, que é o arquivo de configuração da aplicação web.

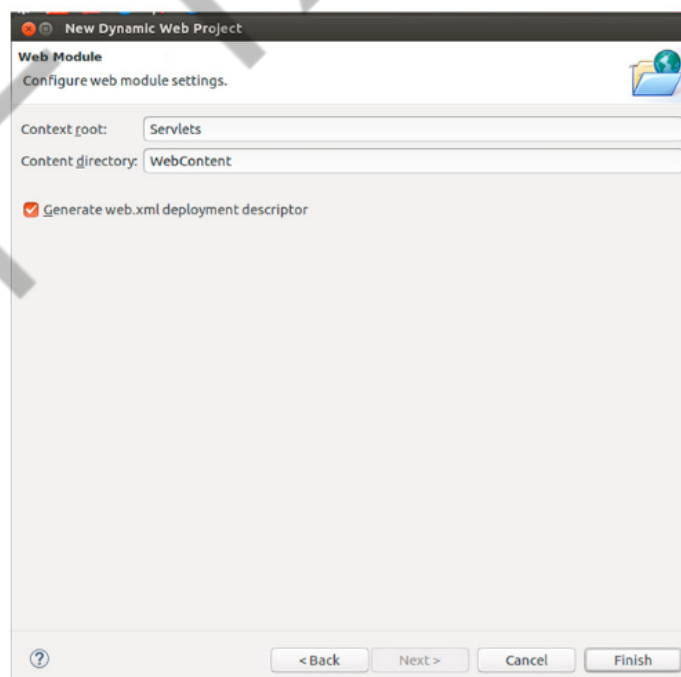


Figura 24 – Criando um projeto web no eclipse – Parte 3
Fonte: Elaborado pelo autor (2017)

Depois de finalizar o processo, observe a estrutura de diretórios que foram criados. Os dois diretórios principais que você precisa identificar são o Java Resources/src, em que ficam os arquivos Java e o diretório WebContent, no qual ficam os arquivos web, como páginas, css, javascript, imagens, fontes etc.

Para o nosso exemplo, vamos criar uma página HTML e um formulário para enviar as informações para uma *Servlet*. Clique com o botão direito do mouse na pasta *WebContent* e escolha a opção New > HTML File (Figura “Criando uma página HTML – Parte 1”).

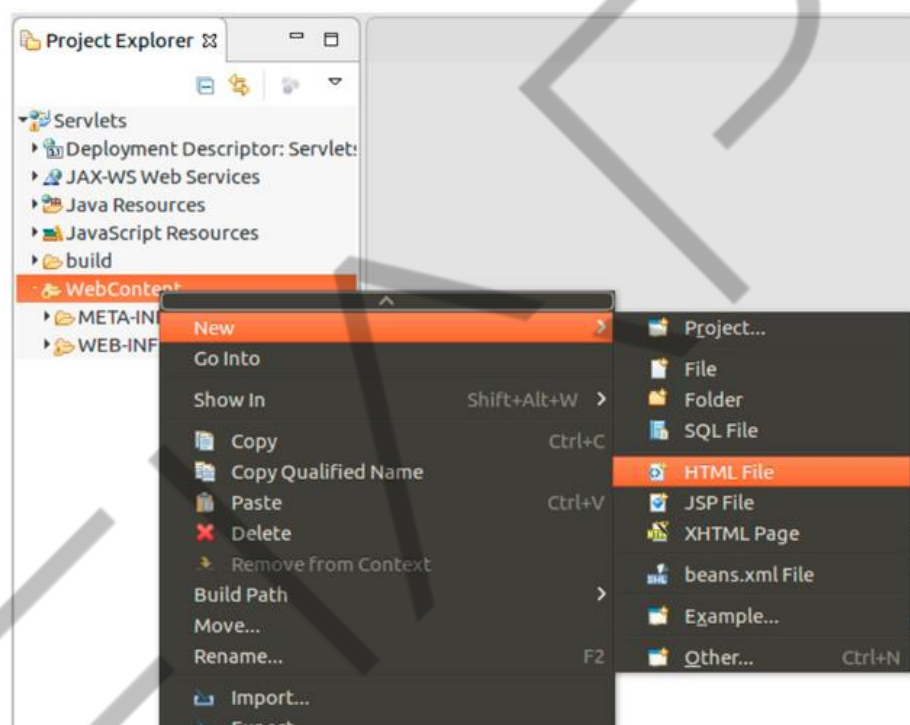


Figura 25 – Criando uma página HTML – Parte 1
Fonte: Elaborado pelo autor (2017)

Agora, dê um nome para a página. Como sugestão, vamos utilizar o nome “index”, porém podemos utilizar qualquer nome que não fará muita diferença nesse momento (Figura “Criando uma página HTML – Parte 2”).

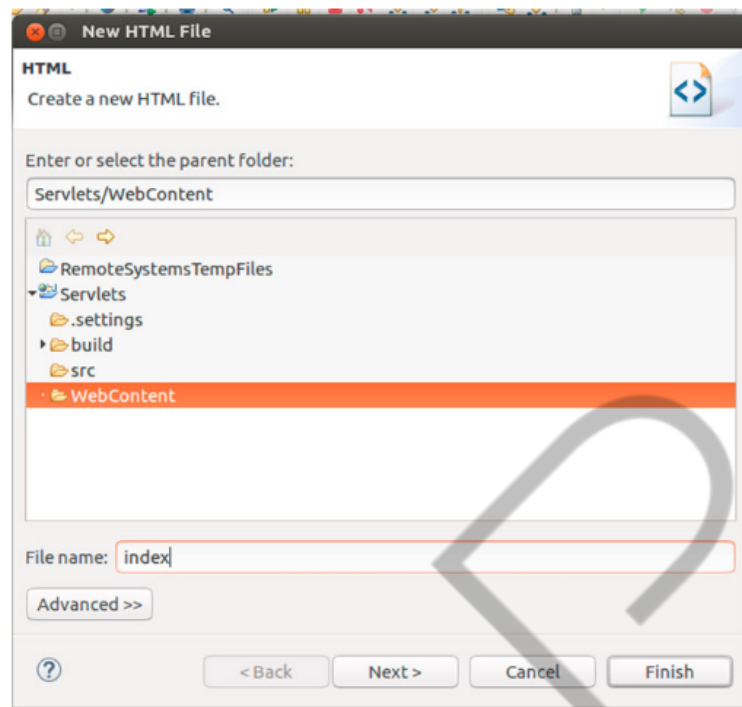


Figura 26 – Criando uma página HTML – Parte 2
Fonte: Elaborado pelo autor (2017)

Vamos implementar um formulário HTML para o cadastro de um produto, os campos serão: nome, quantidade e valor, conforme o código-fonte “Formulário de cadastro de produto”.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>FIAP - Servlets</title>
</head>
<body>
  <h1>Cadastro de Produto</h1>
  <form action="produto" method="post">
    <div>
      <label for="id-nome">Nome</label>
      <input type="text" name="nome" id="id-
nome">
    </div>
    <div>
      <label for="id-qtd">Quantidade</label>
      <input type="text" name="quantidade"
id="id-qtd">
    </div>
    <div>
      <label for="id-valor">Valor</label>
      <input type="text" name="valor" id="id-
valor">
  </form>
</body>
</html>
```

```
</div>
<input type="submit" value="Salvar">
</form>

</body>
</html>
```

Código-fonte 27 – Formulário de cadastro de produto
Fonte: Elaborado pelo autor (2017)

Observe a configuração do formulário, a *action* é “produto” e o método do HTTP é POST. Só para lembrar, a *action* define a URL para a qual serão enviados os valores do formulário. O método POST envia os parâmetros (valores) do formulário no corpo da requisição, já o método GET envia os valores pela própria URL.

Outros pontos importantes são os nomes dos parâmetros. O atributo *name* dos campos define o nome do parâmetro que será enviado para o servidor, esse nome será utilizado na *Servlet* para recuperar as informações inseridas pelo usuário.

Agora, vamos criar a *Servlet* para receber as informações do formulário. Clique com o botão direito do mouse na pasta *src* e escolha a opção New > Servlet (Figura “Criando uma classe *Servlet* – Parte 1”).

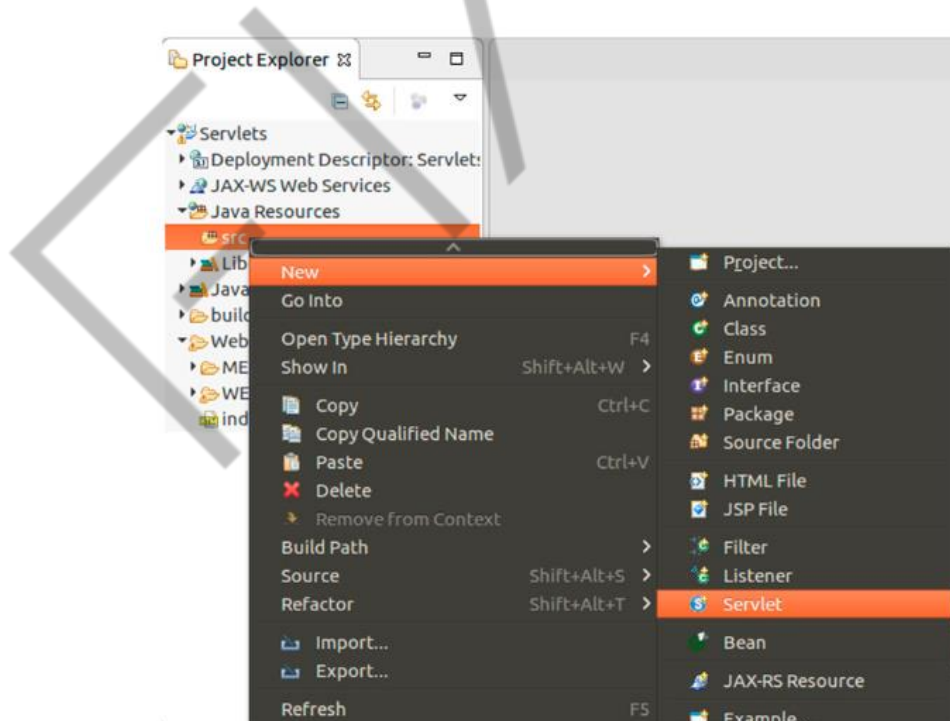


Figura 27 – Criando uma classe *Servlet* – Parte 1
Fonte: Elaborado pelo autor (2017)

Como uma *Servlet* é uma classe Java, defina um pacote (Java package) e um nome, (Class name). Como sugestão, utilizamos o pacote **br.com.fiap.controller** e o

nome de classe **ProdutoServlet** (Figura “Criando uma classe Servlet – Parte 2”). O nome controller utilizado no pacote é referente ao padrão arquitetural MVC (*Model-View-Controller*), logo falaremos mais sobre o MVC. Por ora, vamos começar a utilizar os nomes corretos para ficar mais fácil de entender no futuro. Por padrão, é interessante colocar o nome da classe *Servlet* terminando com *Servlet* (ProdutoServlet) ou *Controller* (ProdutoController). Dessa forma, só de ler o nome da classe já entenderemos a responsabilidade dela no sistema.

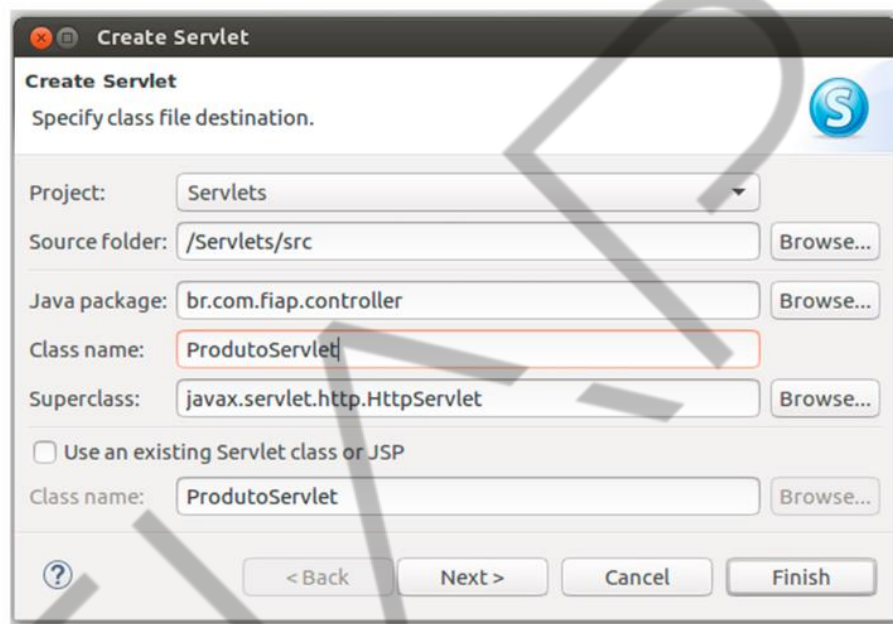


Figura 28 – Criando uma classe *Servlet* – Parte 2

Fonte: Elaborado pelo autor (2017)

Vamos ajustar o código da *Servlet* para recuperar as informações do formulário. O primeiro passo é configurar a *servlet* para atender a *action* (URL) que está configurada no formulário. Dessa forma, o valor do `@WebServlet` (“/produto”) deve ser igual ao do *action* (“produto”) do formulário. Outro detalhe importante é o método configurado no formulário (POST). Assim, precisamos utilizar o método **doPost** da *Servlet* para recuperar os dados.

Depois de “amarrar” o formulário com a *Servlet*, chegou a hora de recuperar os valores enviados. Para isso, utilizamos o método `getParameter(“parametro”)`, da *request*. Lembrando que o valor “parâmetro” deve ser igual ao nome do campo do formulário. Os valores sempre são retornados com o tipo *String*, daí a necessidade de realizar as conversões de dados (*integer* e *double*).

Imprimimos os valores recuperados no console do eclipse para testar. Por fim, queremos exibir uma página de resposta para o usuário. Nesse momento, não vamos utilizar a *servlet* para isso, e sim redirecionar para outra página e enviar os dados para que ela possa exibir as informações.

Para enviar dados para a página, adicionamos atributos no *request* por meio do método `setAttribute("nome","valor")`. O "nome" será utilizado pela página para recuperar o valor. Finalmente, redirecionamos a requisição para a página "sucesso.jsp". Toda essa implementação pode ser vista no código-fonte "Implementação do ProdutoServlet".

```
package br.com.fiap.controller;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.catalina.connector.Request;

@WebServlet("/produto")
public class ProdutoServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {

        //Recuperar os parâmetros do formulário HTML
        String nome = request.getParameter("nome");
        int qtd = Integer.parseInt(request.getParameter("quantidade"));
        double valor = Double.parseDouble(request.getParameter("valor"));

        //Exibe os valores no console do eclipse
        System.out.println(nome + " " + qtd + " " +
            valor);

        //Adiciona atributos no request para exibir na
        página

        request.setAttribute("nomeProduto", nome);
        request.setAttribute("quantidade", qtd);
    }
}
```



```
request.setAttribute("valorProduto", valor);

request.getRequestDispatcher("sucesso.jsp").forward(request, response);
}

}
```

Código-fonte 28 – Implementação do ProdutoServlet

Fonte: Elaborado pelo autor (2017)

Agora precisamos criar a página “sucesso.jsp”. Para isso, clique com o botão direito na pasta *WebContent* e escolha New > JSP File (Figura “Criando uma página JSP– Parte 1”). Vamos precisar de uma página JSP para recuperar as informações enviadas pela *Servlet*. No próximo capítulo, abordaremos os JSPs.

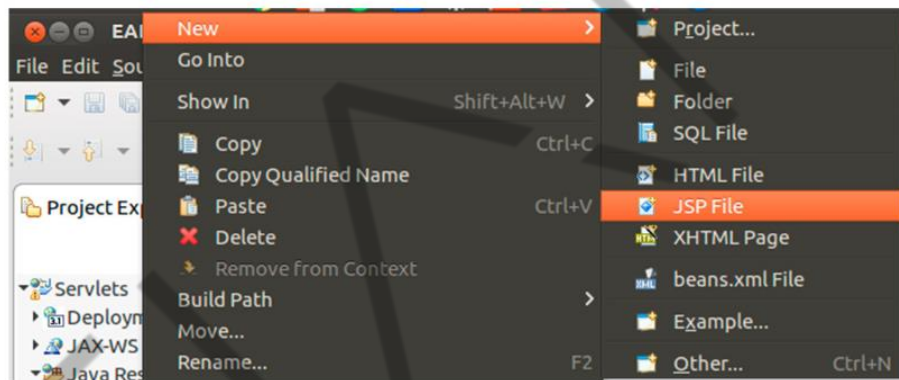


Figura 29 – Criando uma página JSP– Parte 1

Fonte: Elaborado pelo autor (2017)

O próximo passo para criar o JSP é dar um nome a ele (Figura “Criando uma página JSP– Parte 2”). Depois, é só finalizar.

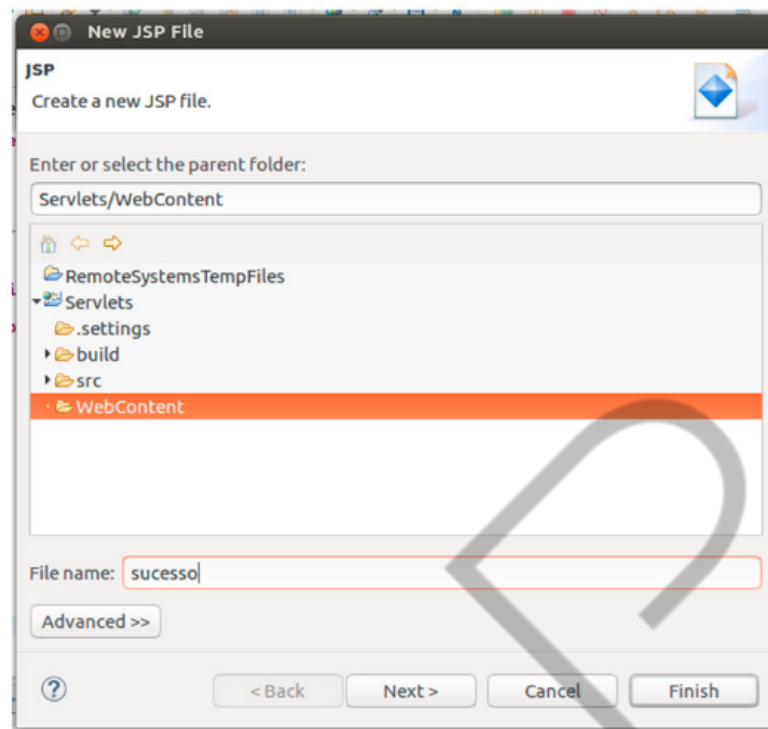


Figura 30 – Criando uma página JSP– Parte 2
Fonte: Elaborado pelo autor (2017)

A implementação do JSP pode ser vista no código-fonte “Página que exibe as informações enviadas pelo formulário”. Utilizamos os nomes dos atributos que foram inseridos na *Servlet* para exibir na página. Os nomes dos atributos devem estar dentro das chaves ({ \$ }). Nos próximos capítulos, falaremos sobre esse comando, chamado *Expression Language*.

```
<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Sucesso!</title>
</head>
<body>

    <p>O Produto foi cadastrado com sucesso! </p>
    <p>${nomeProduto }, ${quantidade }, ${valorProduto
}</p>
```

```
</body>  
</html>
```

Código-fonte 29 – Página que exibe as informações enviadas pelo formulário
Fonte: Elaborado pelo autor (2017)

Chegou o momento de executar o projeto. Clique com o botão direito do mouse na página `index.html` e escolha `Run As > Run on Server` (Figura “Executando a aplicação no Tomcat (servidor) – Parte 1”).

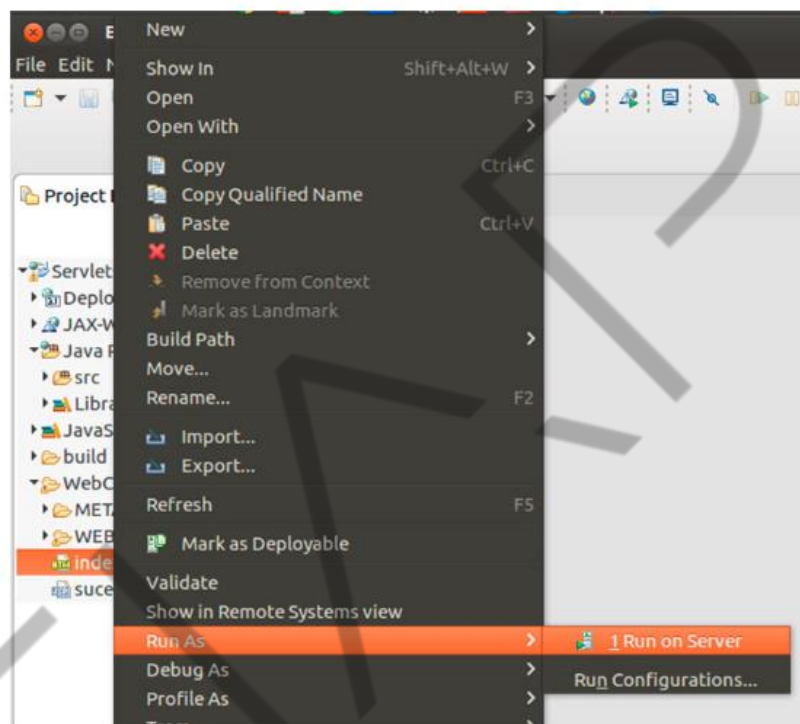


Figura 31 – Executando a aplicação no tomcat (servidor) – Parte 1
Fonte: Elaborado pelo autor (2017)

O próximo passo é escolher o servidor para rodar a aplicação, no nosso caso, utilizaremos o *Tomcat* (Figura “Executando a aplicação no Tomcat (servidor) – Parte 2”).

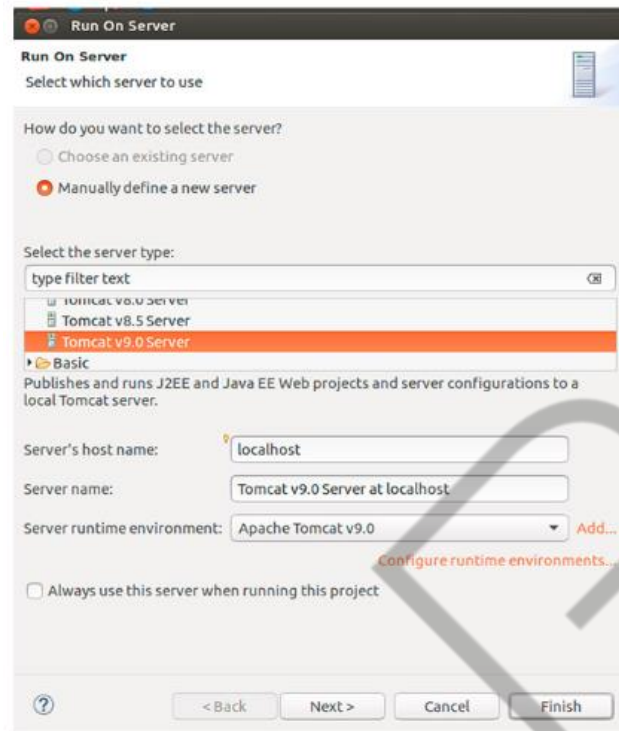


Figura 32 – Executando a aplicação no Tomcat (servidor) – Parte 2
Fonte: Elaborado pelo autor (2017)

Após escolher o servidor, basta finalizar para o *Tomcat* ser inicializado. O resultado da execução pode ser visto na Figura “Resultado da execução da página index.html no servidor”.

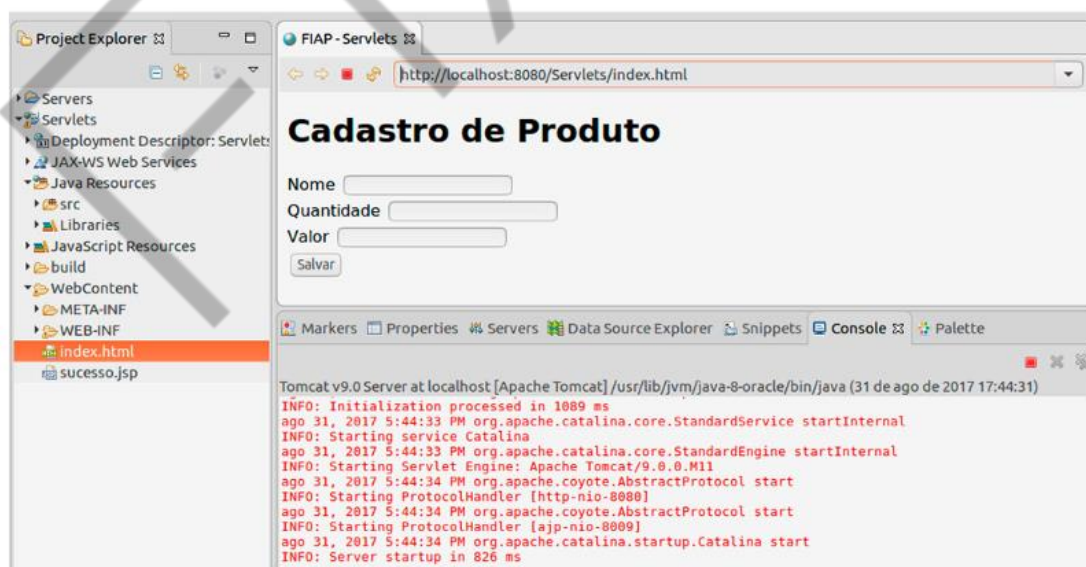


Figura 33 – Resultado da execução da página index.html no servidor
Fonte: Elaborado pelo autor (2017)

Observe que a página foi aberta no navegador interno do eclipse. Se preferir, você pode escolher um outro browser, basta ir em Window > Web Browser e escolher

seu navegador preferido (Figura “Configuração do browser em que será aberta a página”). Também é possível copiando e colando a URL em um outro browser.

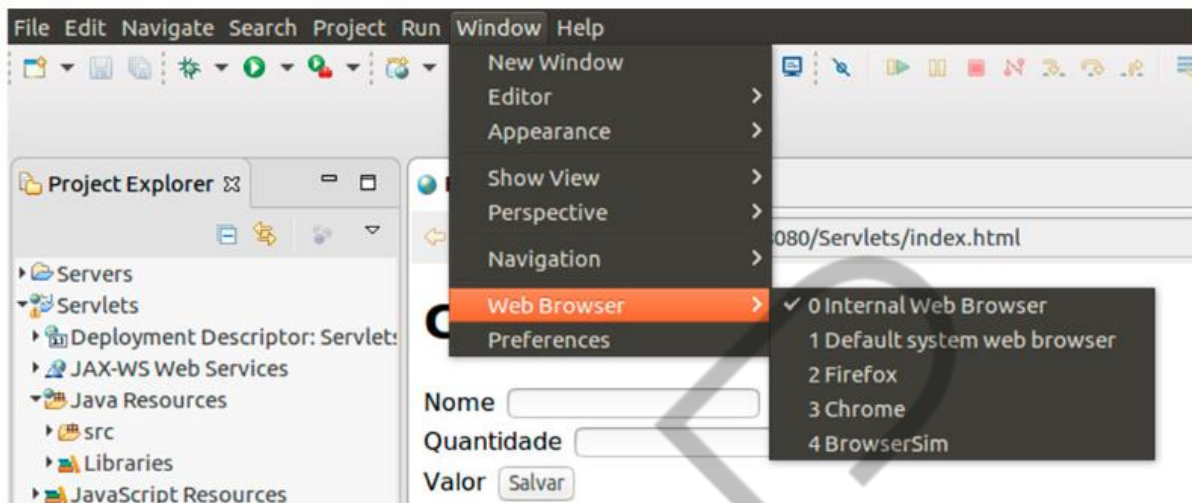


Figura 34 – Configuração do browser em que será aberta a página
Fonte: Elaborado pelo autor (2017)

Vamos realizar o teste, preencha com as informações. Lembre-se de que utilizamos o ponto (.) para separar as casas decimais (Figura “Preenchimento do formulário”).

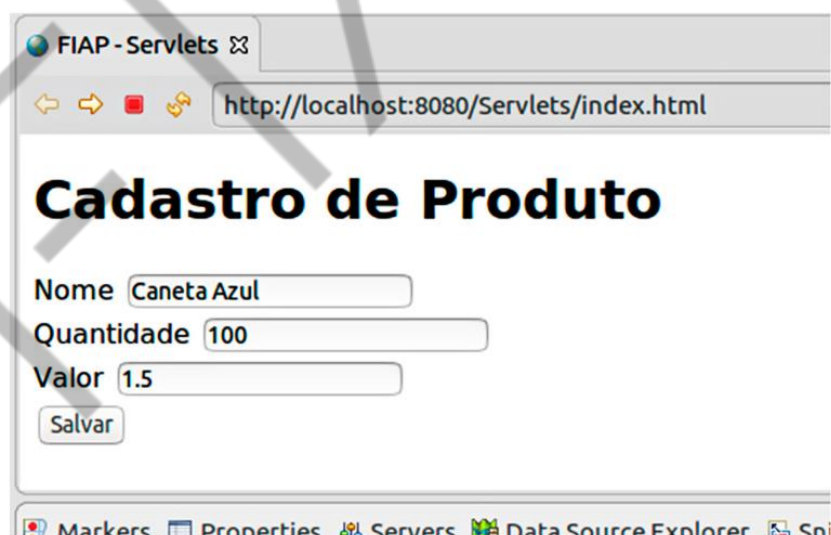


Figura 35 – Preenchimento do formulário
Fonte: Elaborado pelo autor (2017)

Depois de enviar as informações, clicando no botão **Salvar**, os dados serão exibidos no console do eclipse e na página. Observe a URL do browser, ela foi alterada para o valor da *action* do formulário (Figura “Resultado do envio dos dados do formulário”).

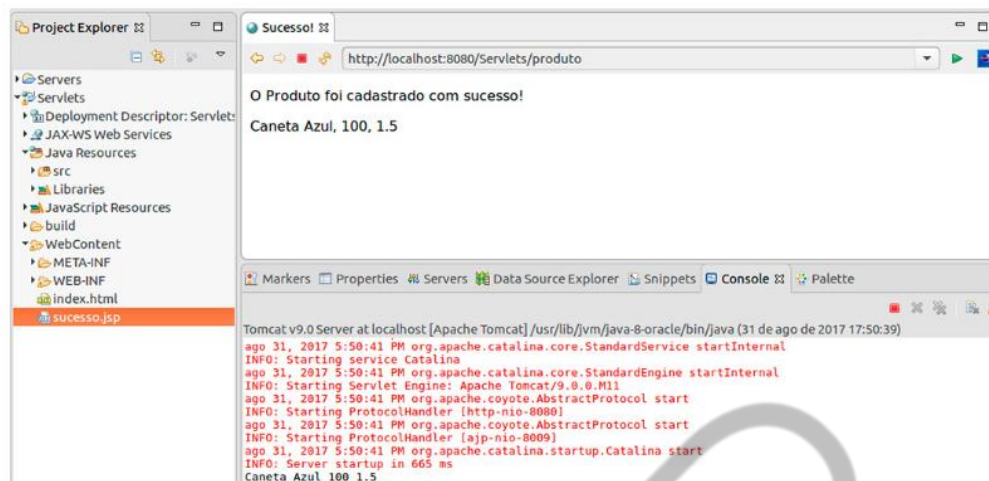


Figura 36 – Resultado do envio dos dados do formulário
Fonte: Elaborado pelo autor (2017)

Vamos fazer outro exemplo, agora com links enviando valores para as *Servlets*. Para isso, vamos adicionar um link na página *index.html*.

```
<a href="produto?codigo=1">Buscar</a>
```

Código-fonte 30 – Link para chamar uma *Servlet*
Fonte: Elaborado pelo autor (2017)

Esse link pode ser adicionado em qualquer lugar na página. Em nosso exemplo, colocaremos logo depois do formulário (Figura “Exibição do link que foi adicionado à página”). Esse link tem o destino para “produto” e envia um parâmetro código com valor 1.



Figura 37 – Exibição do link que foi adicionado à página
Fonte: Elaborado pelo autor (2017)

Para onde será que o link está apontando? Para a *Servlet*. Lembra da configuração do `@WebServlet`? Não são os mesmos valores? Inclusive é o mesmo valor da *action* do formulário. Mas se o link e o formulário “chamam” a mesma *Servlet*, como vamos diferenciá-los?

A resposta é: por meio do método do HTTP. Lembre-se: os links realizam requisições GET, já os formulários podem realizar os dois tipos (GET e POST), assim configuramos o formulário para enviar requisição POST. Dessa forma, quando o link for acionado, o método **doGet** da *Servlet* será chamado. Já quando o formulário for enviado, o método **doPost** é que será acionado. Vamos implementar o método **doGet** da *servlet* (código-fonte “Método doGet da *Serlvet*”).

```
package br.com.fiap.controller;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/produto")
public class ProdutoServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

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

        int                codigo                =
        Integer.parseInt(request.getParameter("codigo"));

        System.out.println("Código: " + codigo);

        request.setAttribute("cod", codigo);
        request.setAttribute("nome",          "Caneta
Vermelha");

        request.getRequestDispatcher("busca-
produto.jsp").forward(request, response);
    }
}
```

```

        protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {

            //Recuperar os parâmetros do formulário HTML
            String nome = request.getParameter("nome");
            int qtd = Integer.parseInt(request.getParameter("quantidade"));
            double valor = Double.parseDouble(request.getParameter("valor"));

            //Exibe os valores no console do eclipse
            System.out.println(nome + " " + qtd + " " +
            valor);

            //Adiciona atributos no request para exibir na
            página
            request.setAttribute("nomeProduto", nome);
            request.setAttribute("quantidade", qtd);
            request.setAttribute("valorProduto", valor);

            request.getRequestDispatcher("sucesso.jsp").forward(request, response);
        }
    }

```

Código-fonte 31 – Método doGet da *Serlvet*
 Fonte: Elaborado pelo autor (2017)

Esse método recupera o parâmetro enviado pelo link (código). Colocamos dois atributos na *request* (código e nome) e encaminhamos a requisição para a página busca-produto.jsp.

Finalmente vamos implementar o JSP para exibir as informações que foram enviadas pela *Serlvet*. Nela, exibimos o código e o nome do produto (código-fonte “Formulário de cadastro de produto”).

```

<%@ page language="java" contentType="text/html;
charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Detalhes Produto</title>
</head>
<body>

```



```
<h1>Detalhes do Produto - ${nome } (Código ${cod  
})</h1>  
  
</body>  
</html>
```

Código-fonte 32 – Formulário de cadastro de produto

Fonte: Elaborado pelo autor (2017)

Vamos testar. Clique no link e veja o resultado (Figura “Resultado da ação de clicar no link”). Não é preciso reiniciar o servidor, mas caso precise, pare e execute novamente o Tomcat.



Figura 38 – Resultado da ação de clicar no link

Fonte: Elaborado pelo autor (2017)

REFERÊNCIAS

DE JESUS, A. C. **Servlets** (Apostila - Notas de Aula). São Paulo: [s.e.], 2014.

SINGH, C. **ServletRequest Interface with example**. 2013. Disponível em: <[https://beginnersbook.com/2013/05/servlet-request-interface/#:~:text=When%20a%20client%20sends%20a,the%20servlet's%20service\(\)%20method.>](https://beginnersbook.com/2013/05/servlet-request-interface/#:~:text=When%20a%20client%20sends%20a,the%20servlet's%20service()%20method.>). Acesso em: 13 set. 2021.

SINGH, C. **ServletResponse Interface**. 2013. Disponível em: <<https://beginnersbook.com/2013/05/servlet-response/>>. Acesso em: 13 set. 2021.