

Curso: Análise e Desenvolvimento de Sistemas – ADS

Ano: 2023/1

Orientação Técnica (OT) – 10

Servlets

Introdução

Servlet é uma das formas de criarmos páginas WEB dinâmicas em Java. Trabalharemos com essa tecnologia apenas nessa OT, mas o conhecimento dela é muito importante, já que a tecnologia que usaremos em seguida (Jersey REST) se baseia nela! Assim, é necessário entender seu funcionamento para entender os demais conteúdos com maior facilidade.

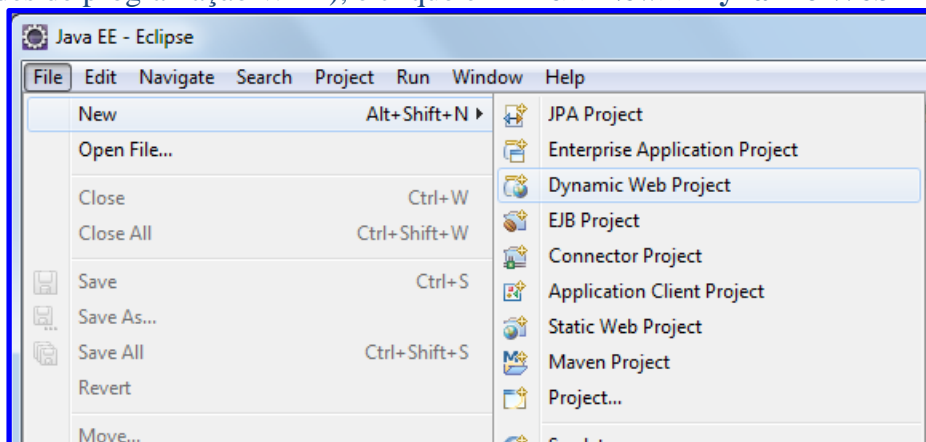
Sabemos que já pesquisou sobre este recurso, então dispensamos neste documento quaisquer definições sobre o mesmo, assim focaremos no desenvolvimento de uma aplicação usando Servlet. Essa aplicação consiste no desenvolvimento de um formulário de cadastro e o uso de Servlet para verificar os dados desse formulário através do servidor. Simples e objetivo, para focarmos no fluxo das ações desde o preenchimento do formulário até o retorno do servidor, mostrando esses dados em outra página. Vamos lá?

Formulário de cadastro

1 – Crie um novo projeto web

Para isso siga os passos a seguir:

Abra a workspace do ProjetoTrilhaWeb (nossa workspace usada até o momento nas atividades de programação WEB), e clique em **File>>New>>Dynamic Web Project**



Será apresentada a janela para a configuração do seu novo projeto web conforme a figura a seguir:

New Dynamic Web Project

Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.

Project name: exemploServlet

Project location:
☒ Use default location
Location: C:\Users\sindirepa\Downloads\WorkspaceGabrielG\exemploServlet

Target runtime:
<None> New Runtime...

Dynamic web module version:
2.5

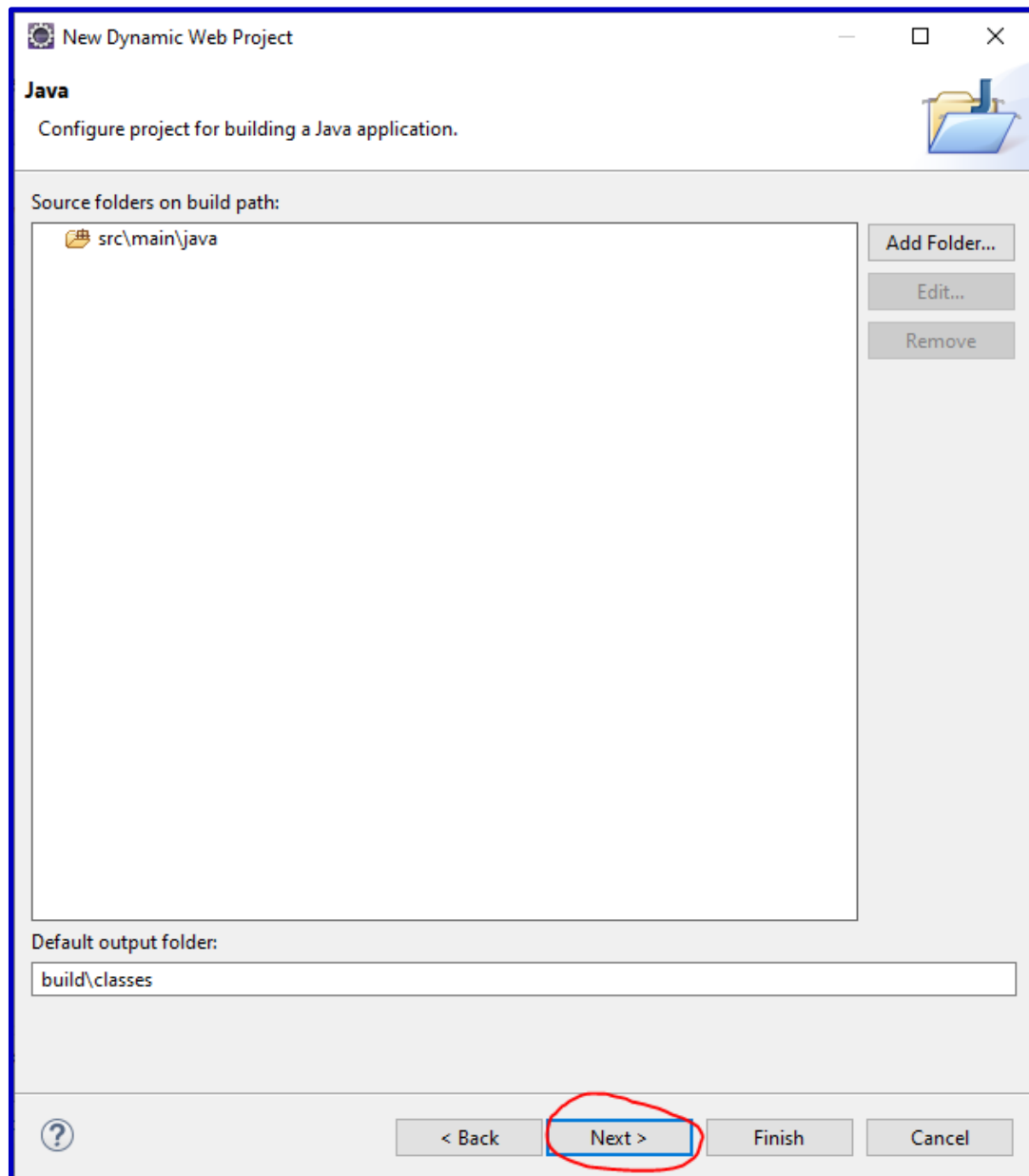
Configuration:
<custom> Modify...
Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR membership:
☐ Add project to an EAR
EAR project name: exemploServletEAR New Project...

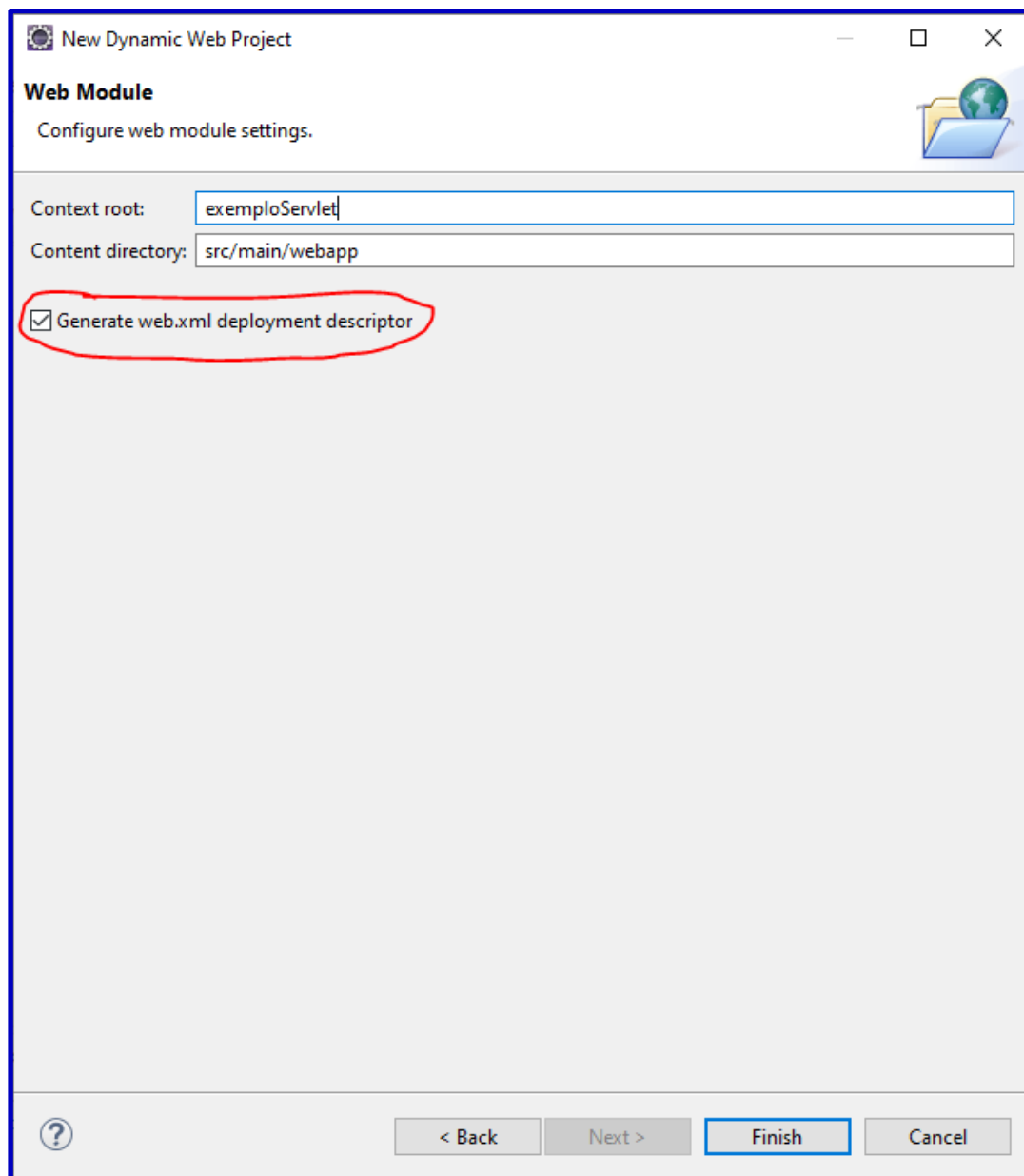
Working sets:
☐ Add project to working sets New...
Working sets: Select...

Navigation:
? < Back **Next >** Finish Cancel

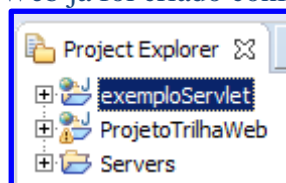
- i. Em **Project name**, informe o nome do projeto: exemploServlet;
- ii. Em **Project location**, deixe assinalado a opção Use default location, para que este projeto seja gravado em sua workspace;
- iii. Em **Dynamic web module version**, escolha a versão 2.5, para que, mais adiante, configuremos manualmente o mapeamento da Servlet no servidor, permitindo que você entenda melhor como uma Servlet deve ser encontrada.
- iv. Clique em **Next**.
- v. Na tela seguinte, deixe tudo como está e clique novamente em **Next**.



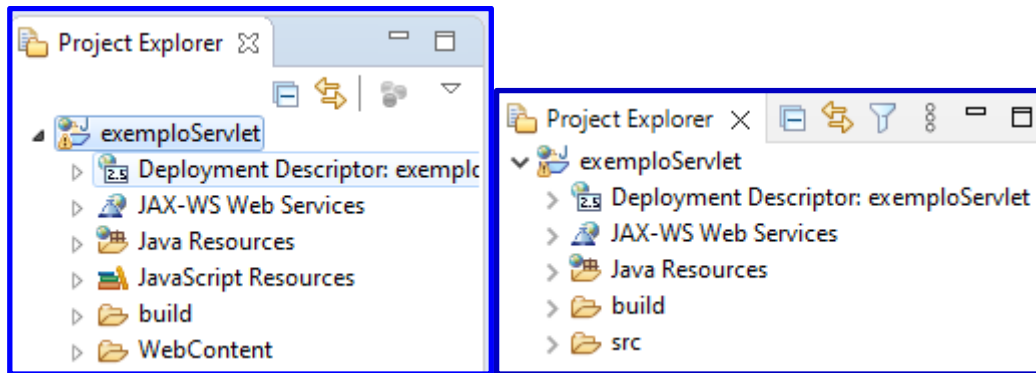
Na próxima tela, vamos alterar uma configuração:



Marque a opção “**Generate web.xml deployment descriptor**” e clique em **Finish**. Se tudo correu bem, seu projeto Web já foi criado conforme a figura a seguir:



Se expandir a estrutura do projeto exemploServlet, verá, dependendo da sua versão do Java, uma das seguintes estruturas:



Não se preocupe com os detalhes ainda, agora o importante é que seu projeto foi criado.

2 – Crie um formulário

O objetivo neste momento é o de criarmos um pequeno formulário, em HTML, para contato com uma pessoa, contendo os campos Nome, Endereço e Telefone.

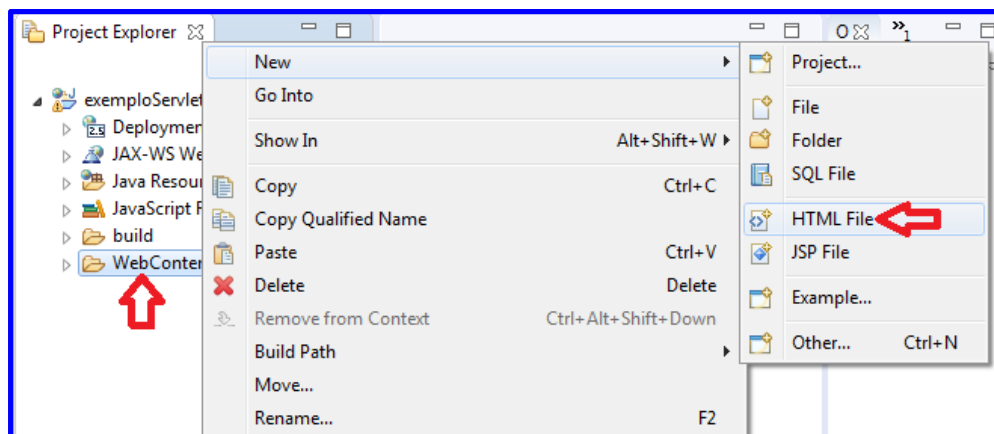
Contato

Nome:

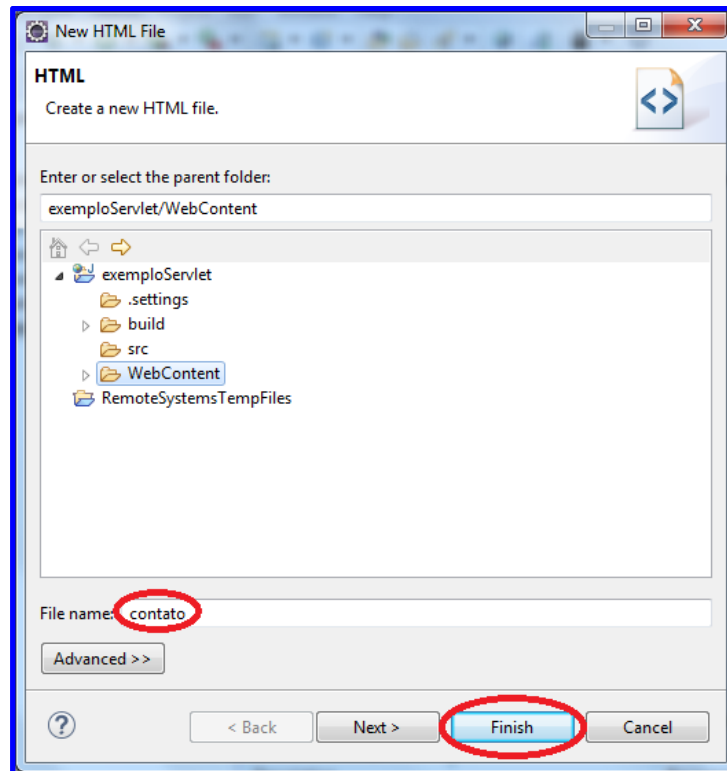
Endereço:

Telefone:

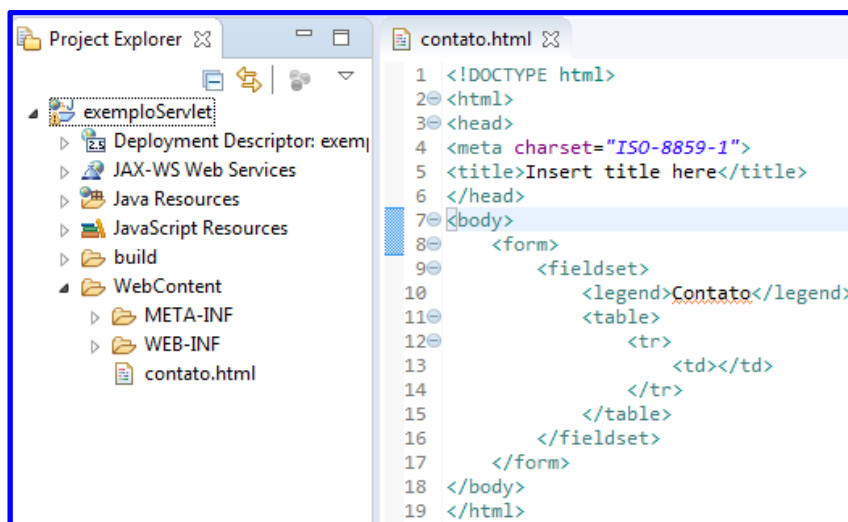
Como já sabe, fazer isso é simples: localize a pasta **WebContent** (Conteúdo Web) no seu projeto. Clique com o botão direito do mouse em cima dela e obterá:



Escolha **HTML File**. Aparecerá a janela para criação do arquivo de programa, informe o nome do arquivo (contato) e clique no botão **Finish**:



Como resultado obterá o arquivo de programa **contato.html**, podendo, a partir de agora, escrever o código para a criação do formulário, utilizando seus bons e velhos conhecimentos em HTML, normalmente como está habituado.



Agora, vamos testar o que foi feito no servidor, mas para isso...

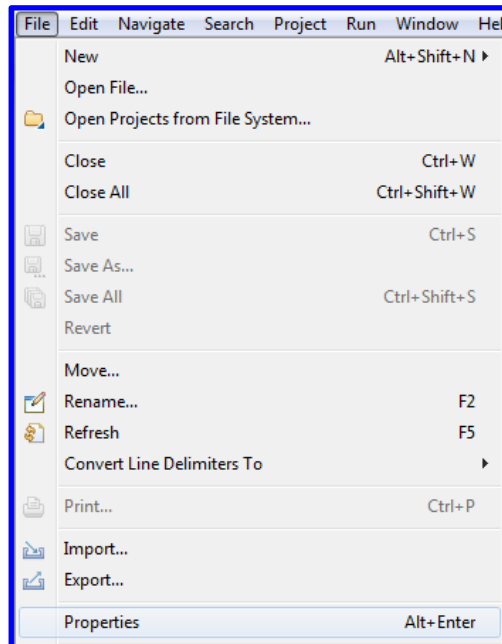
Preparando o projeto para sua execução no servidor

1 - Servidor Apache Tomcat

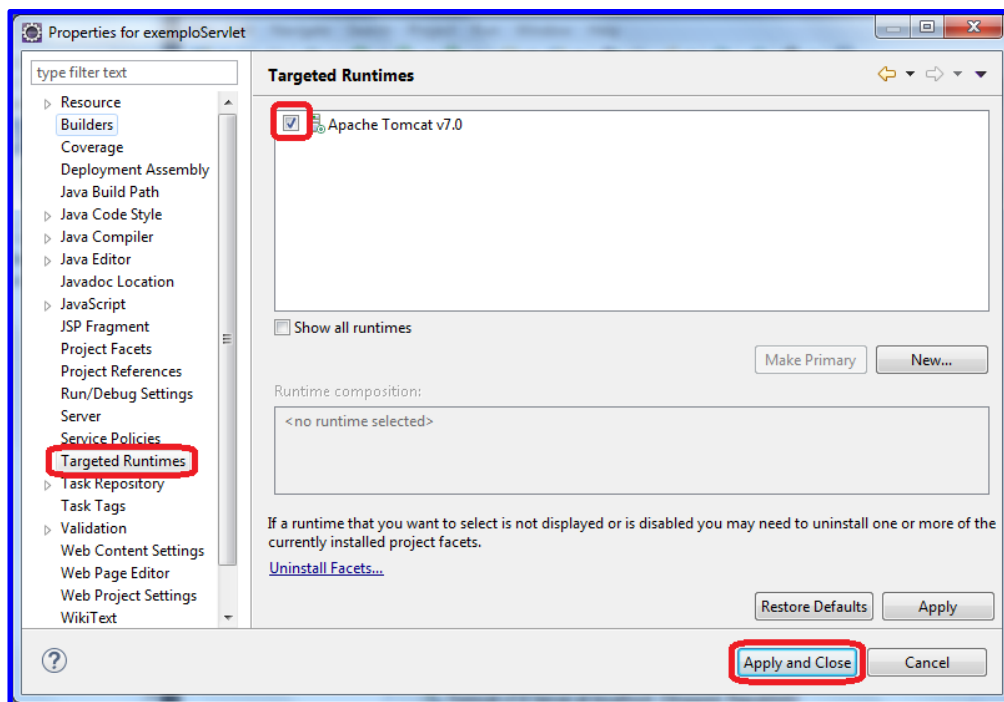
Uma das vantagens de ter criado esse projeto na mesma workspace do projeto anterior, é que já temos uma instância do Tomcat para usar! Para executarmos o projeto novo no Tomcat existente, é só configurarmos isso no projeto.

2 – Especificando para o projeto o ambiente em que este será executado.

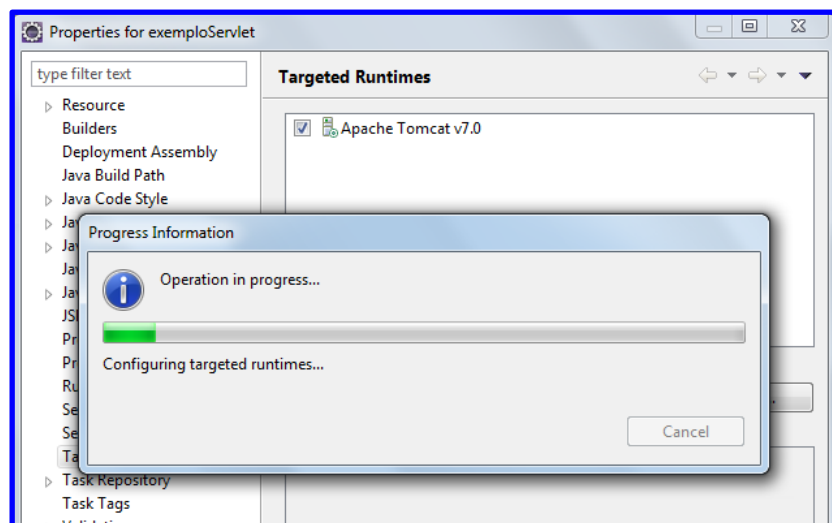
Para isto, clique seleccione no Project Explorer a pasta do projeto (nesse caso exemploServlet) e depois clique no menu **File>>Properties**:



Será apresentada uma caixa de diálogo para que selecione o ambiente em que seu projeto será executado, no caso o servidor Apache Tomcat. Nesta, deve selecionar a opção “**Targeted Runtimes**”, clicar na caixa de seleção **Apache Tomcat v7.0** (ou a versão de seu computador, caso seja diferente), e para confirmar, clicar no botão **OK** ou **Apply and Close**, dependendo de sua versão do Eclipse..



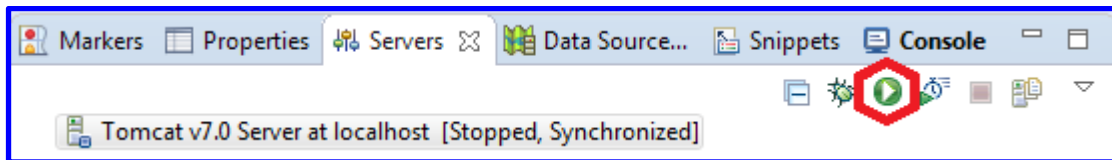
Observe que após clicar neste botão uma configuração é iniciada, estabelecendo para o projeto o ambiente em que este será executado.



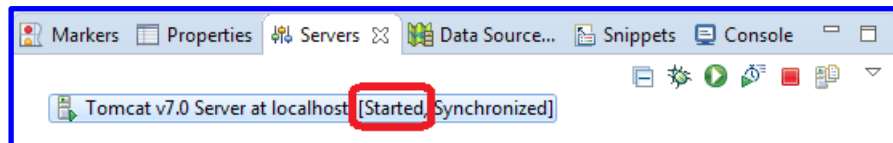
Agora, o melhor: se tudo correu bem, após esta configuração poderemos fazer os testes no servidor!

3 - Testando a aplicação no servidor

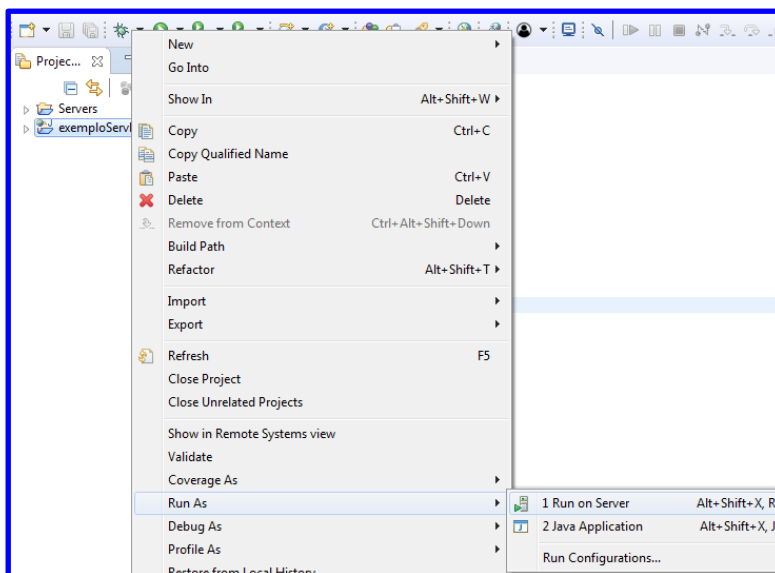
Sempre que quisermos executar algum código nosso no servidor Apache Tomcat, devemos iniciá-lo. Para isso, clique no botão **Start the server**, exibido abaixo:



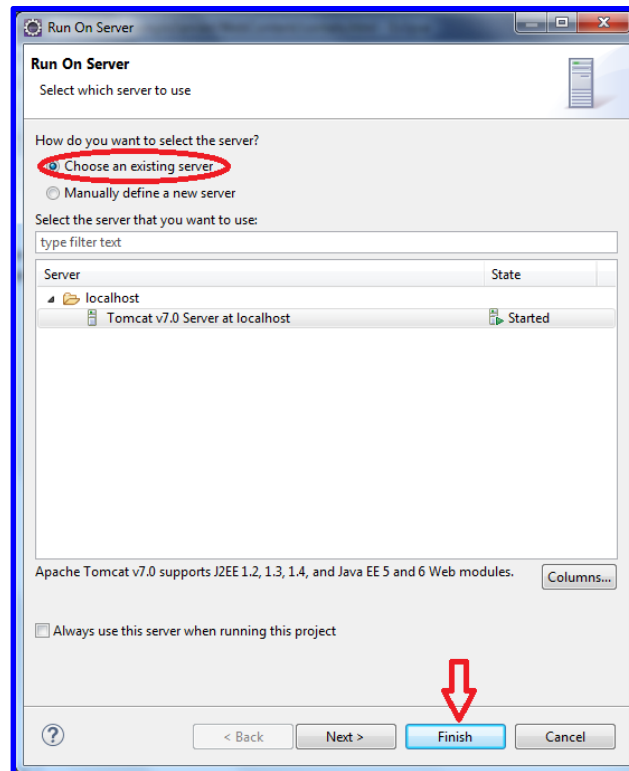
Aguarde, e se tudo der certo, você verá a indicação de que o servidor foi iniciado:



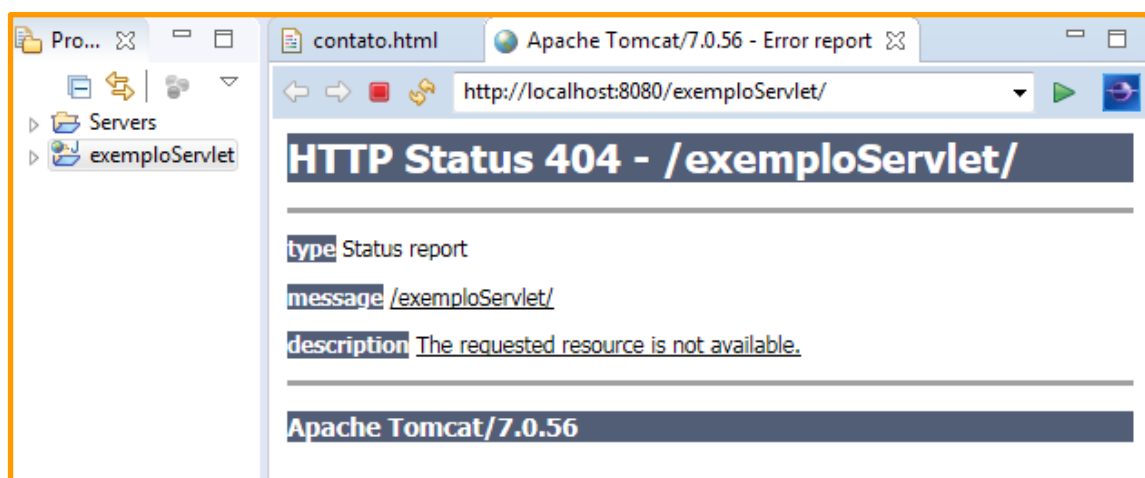
Agora, para executar sua aplicação pode, por exemplo, clicar com o botão direito do mouse no nome do seu projeto e escolher a opção **Run As>>1 Run on Server**, como mostra a figura a seguir:



Em seguida, mantenha as opções conforme a imagem abaixo e clique em **Finish**.

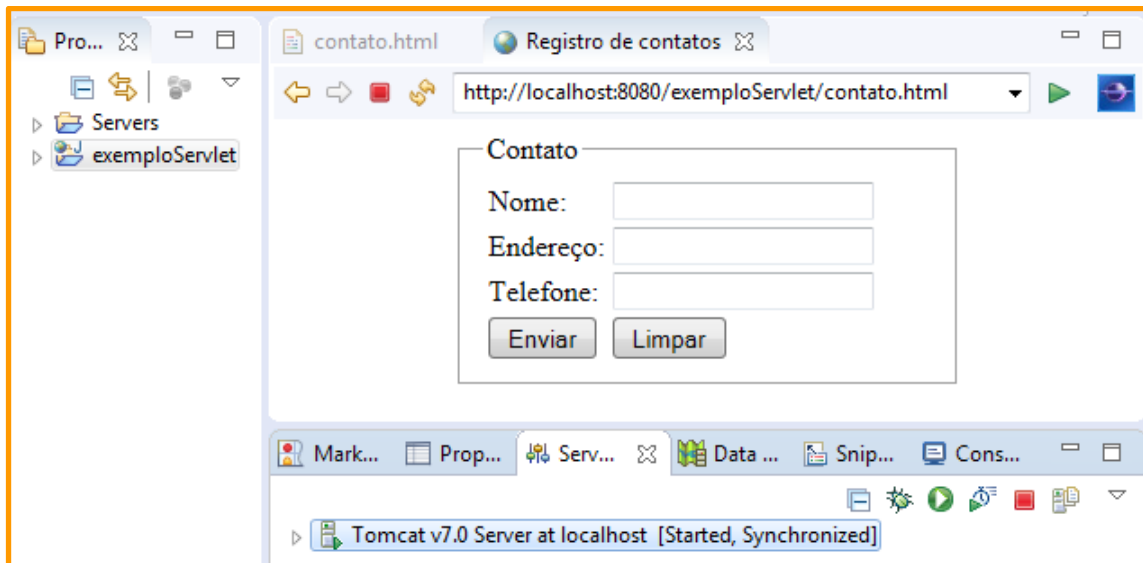


Se for solicitado que reinicie o servidor, faça-o. Por fim, como resultado obterá o seguinte:



ATENÇÃO: Essa aba nova é o navegador web interno do Eclipse, mas recomendamos copiar a URL que apareceu aqui e colá-la no navegador de sua preferência

Mas a vida é uma caixinha de surpresas: temos um erro... Este erro ocorreu porque o arquivo **contato.html** ainda não foi referenciado na URL, então complemente-a conforme abaixo e, mantendo-se na URL, pode, por exemplo, pressionar a tecla “**Enter**”.



Concluindo essa etapa, deve ter percebido que **ainda não utilizamos Servlets**, no momento oportuno faremos isto.

Aprimorando o formulário

Para que possa ambientar-se mais com esta atividade, faça agora o seguinte: **estilize** o formulário com **CSS externo** e desenvolva a **validação** dos campos do formulário em **JavaScript** para que os campos sejam obrigatórios de preenchimento.

Dica: o arquivo CSS deve ficar na mesma pasta do arquivo de programa contato.html, e para chamá-lo, nada de diferente do que já conhece,

```
<link rel="stylesheet" href="XXXXXXXXXX"></link>
```

No caso do JavaScript, faça a validação utilizando uma **função interna** com a tag *script*, pois neste momento não nos interessa muito criar um arquivo JavaScript para isso.

Enviando e recebendo os dados do formulário com Servlet

Vamos agora compreender como enviar e receber as informações de um formulário com o uso de Servlets, para que possamos, por exemplo, validar e armazenar estas informações em uma tabela no banco de dados.

Como percebeu, enquanto trabalhamos no **formulário HTML** estamos lidando com o **lado cliente da aplicação** Java que estamos desenvolvendo. A partir de

agora, **desejamos que os dados registrados neste formulário sejam enviados**, para que no futuro, possam ser **validados e armazenados em uma tabela no banco de dados**. Esta última parte da atividade deve ser realizada no lado servidor da aplicação. Isto significa que a partir de **agora necessitamos utilizar Servlets**, pois estes serão encarregados de **receber a requisição dos dados** do formulário que foram enviados, **processar esses dados** e, por fim, **devolver ao cliente (browser) a resposta** dos dados processados.

Isto significa que, em uma situação normal, uma Servlet seria responsável pelo seguinte:

- Receber os dados do formulário (mais especificamente a requisição);
- Validar e tratar os dados recebidos;
- Gravar estes dados no banco de dados;
- Retornar uma resposta para o lado cliente informando se houve sucesso ou falha nas operações.

Se pensarmos no padrão **MVC** (lembra dele? da trilha de POO? esperamos que sim!) a Servlet atuaria na camada *view*. Obviamente os passos descritos acima não seriam todos implementados na Servlet, cada classe teria a sua responsabilidade em uma funcionalidade de cadastro. A Servlet então estaria entre o lado cliente e servidor, recebendo as requisições, encaminhando às classes responsáveis as tarefas, que por sua vez as executariam e devolveriam os resultados, para que a Servlet possa dar a resposta à requisição feita pelo cliente.

Por questões didáticas, **esta OT só tratará de receber os dados do formulário e imprimí-los** para que possamos compreender bem este mecanismo. As demais tarefas serão realizadas na sequência de nosso estudo, com outras tecnologias.

Agora vamos parar de explicações e vamos ao que interessa. Siga os procedimentos:

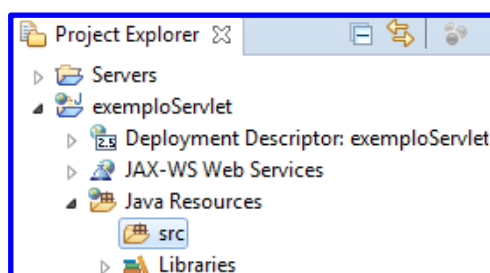
1 – Preparando o formulário

Abra o script do arquivo **contato.html** e no seu formulário, mais especificamente na tag *form*, implemente o atributo **action="insereContato"**. Vale aqui salientar que o atributo *action* indica quem será responsável por receber os dados do formulário assim que eles forem submetidos, com seu valor em algumas linguagens, como PHP, sendo o nome de uma página/arquivo, porém “**insereContato**” não é o nome de uma página, mas sim **o caminho da Servlet** que receberá estes dados.

Importante! Você deve estar se perguntando por qual método estaremos passando os dados, GET ou POST? Por enquanto, não vamos especificá-lo na tag form, mas o mais correto é indicá-lo, e quando não o indicamos, assume-se o método: GET. Aguarde um pouco, mais adiante voltaremos a falar sobre isto.

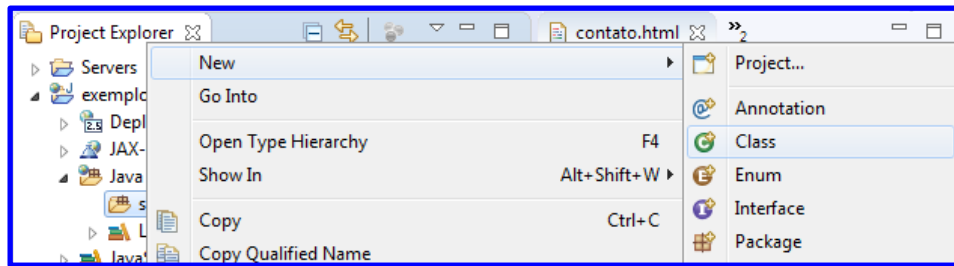
2 – Criando a Servlet

Para isto deve selecionar a pasta **Java Resources** (Recursos Java) e nesta selecionar a pasta **src** (source – origem), conforme figura abaixo:

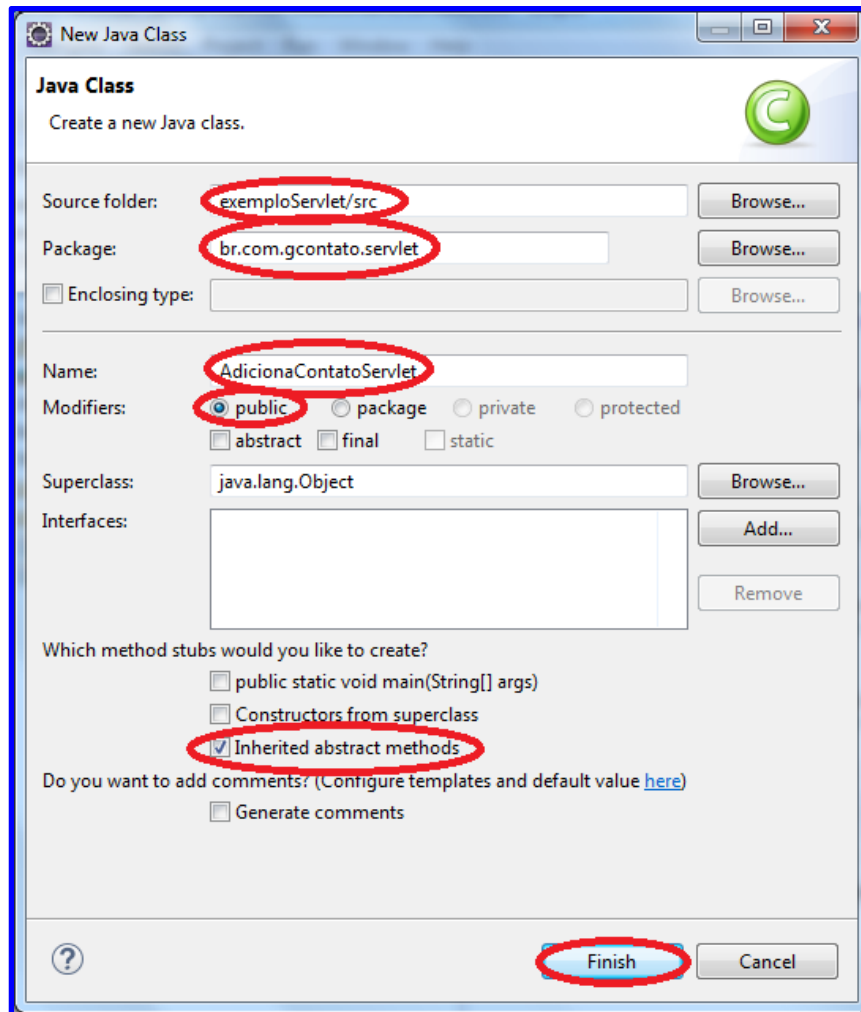


UniSENAI

Continuando o processo de criação da Servlet, deve agora clicar com o botão direito do mouse na pasta src e escolher no menu **New>>Class**, como na figura abaixo:

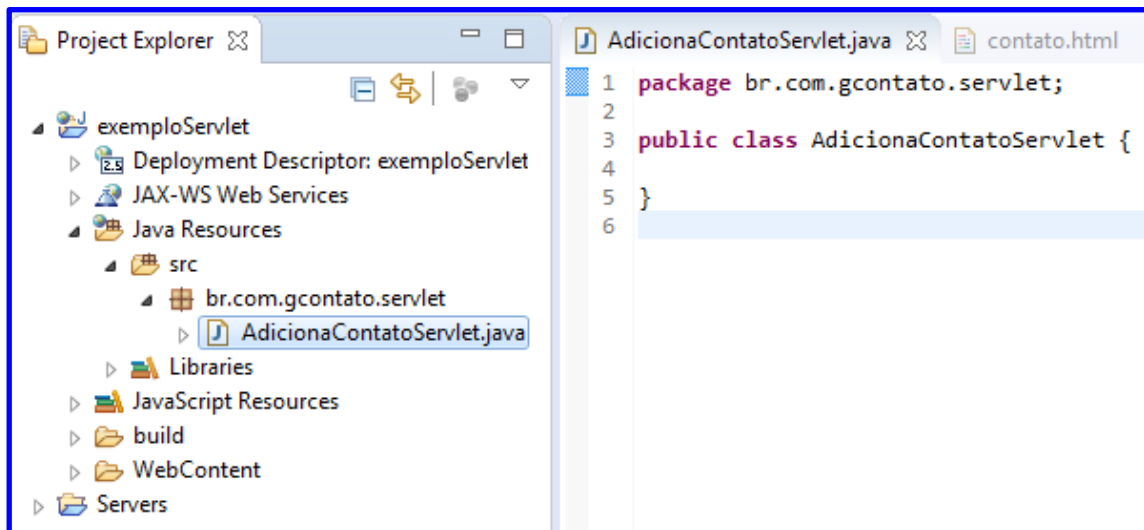


Após clicar na opção **Class**, aparecerá a janela para criação desta classe. Nesta, siga o seguinte:



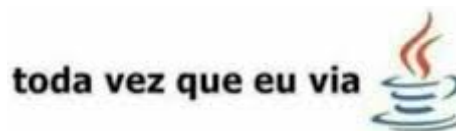
- i. Em **Source folder**, aparece automaticamente o nome da pasta onde está criando a classe;
- ii. Em **Package**, deve informar o nome do pacote onde deseja criar esta classe. Observe que o nome do pacote obedece a estrutura do nome de um domínio web, que deve ser o da sua aplicação, no caso por ser um exemplo optamos por este acima. Mais adiante falaremos mais sobre isto;
- iii. Em **Name**, deve informar o nome da classe deste Servlet que estamos criando;
- iv. Em **Modifiers**, permanece a opção *public*, significando que esta classe é pública e poderá ser acessada por outras classes;
- v. Esta não é uma classe do tipo programa principal, mas sim uma classe Servlet, por isso **não marcamos a opção *public static void main***. Porém deixamos marcada a opção ***Inherited abstract methods***, significando que ela poderá acessar outra classe por herança;
- vi. Clique em **Finish**.

Se tudo correu bem sua classe já foi criada, conforme figura a seguir:



3 - Codificando a Servlet

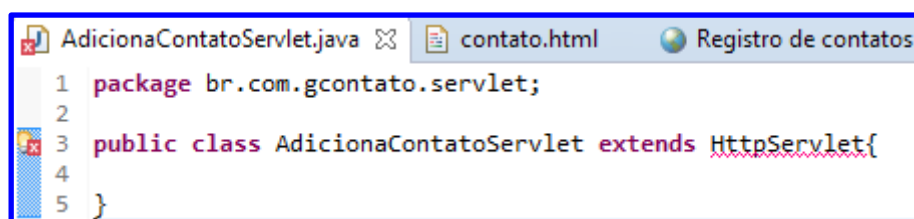
Estava com saudades de programar em Java, né? Como já dizia Sérgio Reis:



*Enfim, chegou a hora de
amiga!*

revermos nossa

Para que ela possa responder as requisições Web que lhe serão solicitadas, ou seja, para fazer com que nosso “Servidorzinho” funcione, torna-se necessário associá-lo por Herança com uma classe Java, a `HttpServlet` que, por sua vez, contém métodos responsáveis por atender requisições e gerar as respostas necessárias. Para isto customize o código da sua classe para o seguinte:



A palavra **extends**, como deve lembrar, significa que estamos **extendendo a classe `HttpServlet`**, ou seja, representa a **associação por Herança** entre as classes **`AdicionaContatoServlet`** (nosso Servlet) e a **classe Java `HttpServlet`**, que assim nos proverá os recursos necessários.

Note que já apareceu erro, é capaz de responder o porquê? Isto mesmo, estamos estendendo uma classe, mas não estamos enxergando ela, por isso o próximo passo é importá-la:


```
AdicionaContatoServlet.java contato.html Registro de contatos
1 package br.com.gcontato.servlet;
2 import javax.servlet.http.HttpServlet;
3 public class AdicionaContatoServlet extends HttpServlet{
4
5 }
```

Não esqueça de salvar. Continuando, precisamos agora **acessar um método** muito importante da classe `HttpServlet` que possibilita ao nosso Servlet **atender e retornar as respostas das requisições** que serão feitas pelo navegador. Trata-se do **método `service`**. A questão é a seguinte: este método possui uma estrutura que nos interessa em parte e outra não, então teremos que **sobrescrever** este método (você pesquisou e fichou sobre sobrescrita de métodos no fichamento paralelo de POO). Isso mesmo, escrever “por cima” de sua escrita na classe `HttpServlet`, informando o que desejamos dele. Isto não significa que não estamos o utilizando, apenas estamos **adaptando ele às nossas necessidades**. Para isto, posicione-se dentro da classe `AdicionaContatoServlet` e escreva o seguinte:

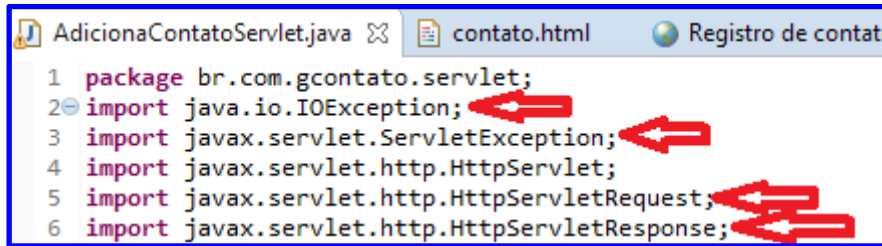
```
AdicionaContatoServlet.java contato.html Registro de contatos
1 package br.com.gcontato.servlet;
2 import javax.servlet.http.HttpServlet;
3 public class AdicionaContatoServlet extends HttpServlet{
4
5     @Override
6     protected void service(HttpServletRequest request,
7                           HttpServletResponse response)
8         throws ServletException, IOException{
9
10    }
11
12 }
```

Destaque para o **@Override**, que indica que estamos **sobrescrevendo** o método `service` da classe mãe `HttpServlet`.

Repare que o método **`service`** recebe dois objetos, **`request`** e **`response`**, que representam, respectivamente, a **requisição** feita pelo usuário e a **resposta** que será criada por nós e enviada ao lado cliente.

Nesse exemplo, temos apenas a intenção de demonstrar a criação e utilização de um Servlet, assim o método não será implementado com nenhuma estrutura lógica avançada, apenas exibirá os dados preenchidos no formulário. Em breve, faremos a implementação nesse método que permitirá isso.

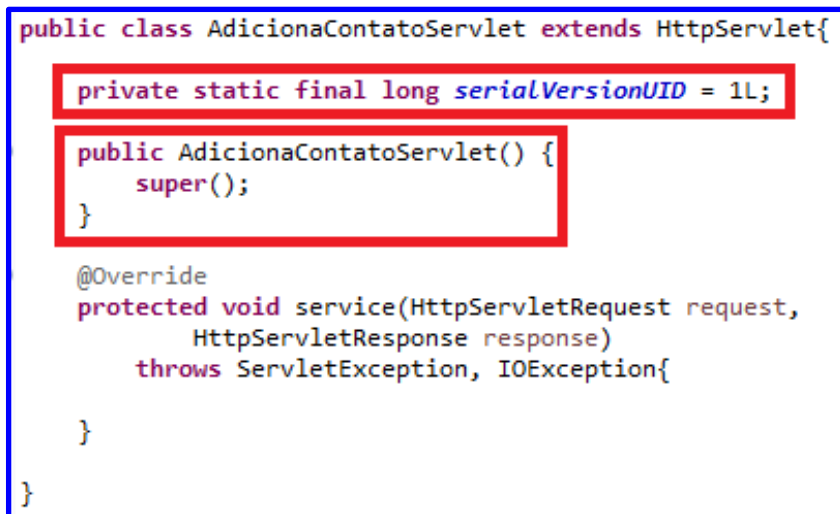
Antes, para resolver os erros atuais, vamos importar as demais bibliotecas Java:



```
1 package br.com.gcontato.servlet;
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
```

Note que importamos mais 4 bibliotecas de classes Java além daquela que já havíamos importado.

Agora, faça a seguinte implementação na Servlet:



```
public class AdicionaContatoServlet extends HttpServlet{
    private static final long serialVersionUID = 1L;

    public AdicionaContatoServlet() {
        super();
    }

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

Vamos entender cada parte nova:

- **private static final long serialVersionUID = 1L;** – trata-se de uma **constante** que serve para gravar o estado de uma classe Java e mandar para algum lugar, por exemplo, para um arquivo, ou até para a Internet. A Java pega os campos da sua classe e os grava. Desta forma, consegue-se manter a versão das suas classes atualizadas. Isso é algo um pouco complexo para esse momento, então oportunamente estaremos detalhando mais e melhor esta constante.
- Abaixo criamos um método **construtor**, que normalmente serve para inicializar os atributos de uma classe quando esta for instanciada. Sobre isto também oportunamente estaremos detalhando mais e melhor, até porque neste momento não estaremos fazendo uso de seus benefícios.

```
public AdicionaContatoServlet() {
    super();
}
```

Continuando a codificação...

```

8 public class AdicionaContatoServlet extends HttpServlet{
9
10     private static final long serialVersionUID = 1L;
11
12     public AdicionaContatoServlet() {
13         super();
14     }
15
16     @Override
17     protected void service(HttpServletRequest request,
18         HttpServletResponse response)
19         throws ServletException, IOException{
20
21     }
22
23     protected void doGet(HttpServletRequest request,
24         HttpServletResponse response)
25         throws ServletException, IOException {
26
27     }
28
29     protected void doPost(HttpServletRequest request,
30         HttpServletResponse response)
31         throws ServletException, IOException {
32
33     }
34
35 }

```

A explicação dessa parte nova é simples: esses dois métodos serão utilizados quando se desejar tratar a passagem de dados, respectivamente, pelos métodos GET e POST.

Agora vamos implementar o método que receberá os dados do formulário Contato. Aqui cabe uma observação importante: Lembre-se que, como não implementamos o atributo “**method**” na tag *form*, o método de passagem de dados assumido é GET.

Então, chega de conversa e vamos ao que interessa! A seguir, implementaremos o método em nossa Servlet que receberá e imprimirá, somente, o nome da pessoa, pois os demais campos será você que implementará!

Primeiro, em nossa classe Servlet, faça o seguinte:

- a) Implemente a importação da biblioteca `java.io.PrintWriter`, para que o Servlet possa enviar para o cliente a impressão dos dados capturados do formulário.

```

package br.com.gcontato.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

- b) Implemente o método que será o responsável por capturar o nome do contato, seguindo o que está dentro do quadrado vermelho:

```

17     @Override
18     protected void service(HttpServletRequest request,
19         HttpServletResponse response)
20         throws ServletException, IOException{
21
22         PrintWriter out = response.getWriter();
23         String nome = request.getParameter("nome");
24         out.println(nome);
25     }
26 }

```

Perceba que agora, realmente, estamos sobrescrevendo o método `service` da classe `HttpServlet` com códigos nossos. Segue comentário sobre as 3 instruções:

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

Para que o usuário veja algo na página web, é preciso **construir a resposta que a Servlet enviará para o cliente**. Isto será feito a partir da criação do objeto **out**, que representa a saída a ser enviada ao usuário por meio do método `getWriter()` do objeto **response**. E a partir disso utilizamos um **PrintWriter**, possibilitando a impressão da resposta ao usuário no corpo da página HTML.

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

Com essa linha, recebemos o valor do campo “nome” do formulário de contato e o armazenamos na variável nome. É importante ressaltar que o `request.getParameter()` **deve receber como parâmetro o valor do atributo “name”** de algum dos campos do formulário. Veja abaixo:

```
<tr>
  <td>Nome:</td>
  <td><input type="text" name="nome"></td>
</tr>
```

```
out.println(nome);
```

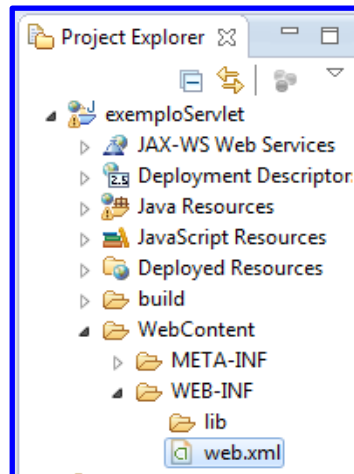
Esta linha é responsável por exibir o valor da variável `nome`, que contém o que o usuário digitou no campo `nome`, como explicado acima.

4 – Mapeando a Servlet para uma URL

Ainda falta um procedimento antes de testarmos a eficiência do nosso código: devemos agora mapear a nossa Servlet para uma URL, assim poderemos executar nossa aplicação no Cliente (Browser).

Agora mão na massa:

Localize a pasta **WEB-INF** e nesta o arquivo **web.xml** como na figura abaixo:



Dê um duplo clique no arquivo **web.xml** para abri-lo para edição e siga os passos indicados.

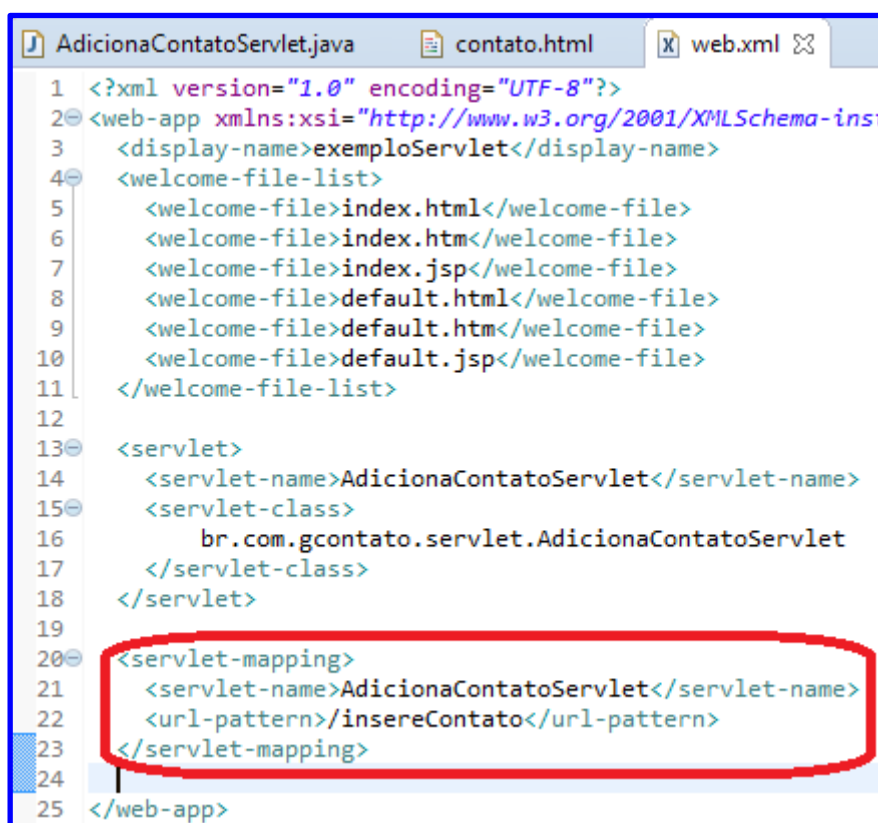
Definindo a Servlet na tag **<servlet>**:



ATENÇÃO: Se não ver o arquivo do mesmo modo que mostrado na imagem acima, clique em **Source**, como indicado na imagem abaixo:



Em seguida, vamos mapear nossa servlet para a URL **/insereContato**. Perceba abaixo que isso acontece dentro da tag **<servlet-mapping>** (mapeamento de servlets) e que você tem que indicar que está referenciando àquela servlet que definimos logo acima, por isso passamos o mesmo **<servlet-name>** para o mapeamento.



Assim, podemos ver que são necessários dois passos para mapear uma servlet para uma URL:

1. Definir o nome e classe da servlet;
2. Definir a URL da servlet, usando seu nome.

Com isso, nossa servlet pode ser acessada por meio da seguinte URL:

localhost:8080/emploServlet/insereContato

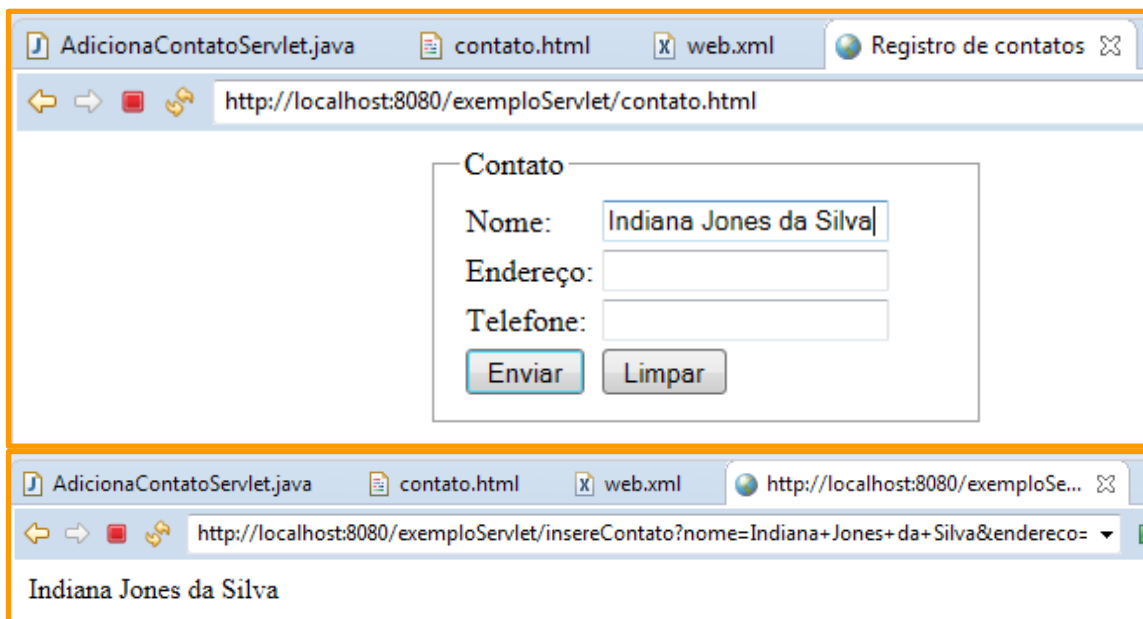
Note também que em

`<url-pattern>/insereContato</url-pattern>`

temos o valor do atributo **action do formulário de contato**, que, no caso, refere-se à Servlet que recebe os dados do formulário e fornece a resposta para o cliente (Browser) mostrando o nome do contato.

5 – Executando a aplicação

Agora sim! Execute a aplicação, a partir do programa do formulário contato.html e ao clicar no botão Enviar, a página de resposta apresenta o nome do contato informado a partir do formulário:



Conseguiu entender a dinâmica do funcionamento da aplicação?

De orientações era isto, agora você trabalhará um pouco para que possa aprender melhor e mais.

Atividades propostas

1. **Melhore a saída de dados** com um título e as informações recebidas mais organizadas. (Informação: A instrução `out.println()` pode receber como parâmetro tags HTML...)
2. Implemente na Servlet o **recebimento e impressão dos demais campos**;
3. Como você viu, implementamos as ações na Servlet dentro do método `service`, certo? Com base nisso, responda às seguintes perguntas:
 - a. Porque tudo funcionou se fizemos o envio dos dados via GET e nossa implementação está no método `service` e não no `doGet`?
 - b. Como você faria a impressão dos dados utilizando os outros métodos que criamos na Servlet: `doGet()` e `doPost()`?

Após finalizar a OT, comunique um orientador para novas instruções.

Fontes:

<http://www.naoentreaki.com.br/5501987-java-e-vida.htm>