Disciplina: Programação Cliente-servidor

Aula 6: Servlet e JSP

Apresentação

Após estudar as diversas tecnologias disponíveis para a construção de interfaces para o cliente, é necessário observar o funcionamento dos servidores e tecnologias associadas, dentro de um ambiente Web completo.

A linguagem Java, com boas características no que se refere a portabilidade e conectividade, é uma opção muito aceita no mundo corporativo para a programação do lado servidor.

Contando com o Web Server Tomcat, o ambiente Java para Web trabalha principalmente com os componentes Servlet e JSP (Java Server Pages), mas outras tecnologias podem ser utilizadas, inclusive com a substituição do Tomcat por um Application Server com suporte a EJB (Enterprise Java Beans), como GlassFish ou JBoss, os quais acrescentam a camada EJB, mas utilizam o próprio Tomcat como container Web, funcionando como módulo interno.

Objetivos

- Explicar o funcionamento do ambiente servidor Java para Web;
- Utilizar a tecnologia de Servlets na construção de páginas dinâmicas;
- Utilizar a tecnologia JSP na construção de páginas dinâmicas.

Fonte: Cody Barnes / Unsplash

Plataforma Java

Java é uma linguagem de programação **orientada a objetos** desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Hoje em dia a linguagem está sob o controle da Oracle, após a mesma ter comprado a Sun.

Desde a sua concepção, o Java trouxe alguns princípios básicos que norteiam a sua própria evolução, destacando-se a preocupação de executar em **múltiplas plataformas** e garantir elementos de **conectividade**.

A linguagem Java utiliza um artifício que permite a execução de seus programas em qualquer plataforma: uso de **máquina virtual**. Com esta abordagem, os programas em Java são **compilados**, não podendo ser classificados como interpretados, mas sem gerar executáveis para o Sistema Operacional, devendo ser executados pela máquina virtual criada para cada plataforma.

Fonte: Lauren Mancke / Unsplash

Nós também temos uma grande facilidade para efetuar a conexão com outros sistemas em rede com o uso de Java, pois ele traz uma extensa biblioteca de componentes para a criação de **Sockets**, conexão **HTTP** e **FTP**, criação de clientes e servidores de **e-mail**, entre diversos outros, o que garante a premissa básica de conectividade da linguagem.

Outra característica inovadora no lançamento do Java, rapidamente adotada por outras linguagens, foi a inclusão de um coletor de lixo (**garbage collector**), o qual efetua a desalocação de memória de forma automática.

Sintaxe Java

Quase tudo em Java é baseado em objetos. Basicamente, apenas os tipos nativos são considerados de forma diferente, mas para cada tipo nativo existe uma ou mais classes **Wrapper**.

Podemos observar, no quadro seguinte, os principais tipos nativos do Java.

Tipo Nativo	Wrapper	Descrição do Tipo
byte	Byte	Inteiro de 1 byte
short	Short	Inteiro de 2 bytes
int	Integer	Inteiro de 4 bytes
long	Long	Inteiro de 8 bytes
char	Character	Caracteres ASCII
float	Float	Real de 4 bytes
double	Double	Real de 8 bytes
boolean	Boolean	Valores booleanos true ou false

As variáveis aceitam diferentes formas de declaração e inicialização, como podemos observar a seguir:

```
int a, b, c;
boolean x = true; // Variável declarada e inicializada
char letra = 'W';
String frase = "Teste";
double f = 2.5, g = 3.7;
```

Para programas simples não há necessidade do uso extensivo de classes Wrapper, mas é aconselhável utilizá-las na criação de sistemas que usem tecnologias de Web Service, ou frameworks de persistência, entre outros.

As linguagens Java e JavaScript descendem da linguagem C; por esse motivo apresentam muitas semelhanças, mas também reservam algumas diferenças.

Os operadores aritméticos, relacionais e lógicos são basicamente os mesmos para as duas linguagens, porém no Java existe uma preocupação maior com o tratamento de elementos binários.

Operador Binário	Operação
&	And (E) binário

|	Or (ou) binário
^	Xor (ou-exclusivo) binário

Vamos observar, a seguir, a estrutura geral de um programa Java.

```
public class Exemplo001 {
  public static void main(String args[ ]) {
    System.out.println("Alo Mundo");
  }
}
```

A linguagem Java é completamente orientada a objetos, e a classe que contém o método estático **main** pode ser executada como um programa de linha de comando. No exemplo seria impressa a frase "Alo Mundo".

As estruturas de decisão e repetição também são equivalentes àquelas encontradas no JavaScript, com algumas diferenças no uso do **for**, pelo fato de o Java ser fortemente tipado.

Vamos observar um exemplo de uso do for a seguir.

```
public class Exemplo002 {
  public static void main(String[] args) {
    // Calculo do valor médio da sequencia y = f(x) = x * x
    // Media = Somatorio dos valores / quantidade
    // Limites 1 a 5
    double soma = 0.0;
    for(int x=1; x<=5; x++)
      soma += Math.pow(x, 2);
      // eleva x a potência 2 e acumula
    System.out.println(soma/5);
  }
}
```

Neste exemplo temos o cálculo do valor médio de uma sequência entre os limites de 1 a 5, considerando a função $\mathbf{y = x^2}$. Foi utilizado o método **Math.pow**, que eleva um número a uma potência qualquer para calcular o quadrado de x.

Executando o programa, teremos a impressão do valor **11**, o que corresponde exatamente ao valor médio, dado por

$$\mathbf{(1 + 4 + 9 + 16 + 25) / 5.}$$

Como a linguagem Java é completamente orientada a objetos, a análise de sua sintaxe exige o conhecimento acerca da definição de classes, com seus atributos e métodos.

Vamos observar a aplicação destes conceitos no exemplo seguinte. Para este exemplo serão necessários dois arquivos, um para Pessoa e outro para Exemplo003.

```
public class Pessoa {
  public String nome;
  public String telefone;
  public void exibir(){
    System.out.println(nome+"::"+telefone);
  }
}
public class Exemplo003 {
  public static void main(String[] args) {
    // Instanciando os objetos p1 e p2
    Pessoa p1 = new Pessoa();
    Pessoa p2 = new Pessoa();
    // Preenchimento dos atributos dos objetos p1 e p2
    p1.nome = "João";
    p1.telefone = "1111-1111";
    p2.nome = "Maria";
    p2.telefone = "2222-2222";
    // Chamada ao método exibir em p1 e p2
    p1.exibir();
    p2.exibir();
  }
}
```

Conforme podemos observar, o primeiro arquivo (Pessoa.java) contém a classe Pessoa, a qual define os atributos nome e telefone e o método exibir.

No outro arquivo (Exemplo003.java) são instanciados os objetos **p1** e **p2** com o uso do operador **new**, responsável por alocar estes objetos e, acompanhando a sequência de comandos, teremos o preenchimento dos atributos nome e telefone para cada um dos objetos, e a chamada ao método exibir de cada um deles. Teremos como resultado final a saída seguinte.

```
João::1111-1111
Maria::2222-2222
```

Note que todos os elementos de **Pessoa** foram declarados como públicos. Na orientação a objetos, as classes definem estruturas fechadas, nas quais o acesso a seus atributos e métodos deve ser controlado, e para tal iremos contar com três níveis de acesso:

Público (public)

Permite que qualquer um acesse o atributo ou método;

Privado (private)

Não permite acessos externos, sendo utilizado apenas na programação interna da classe;

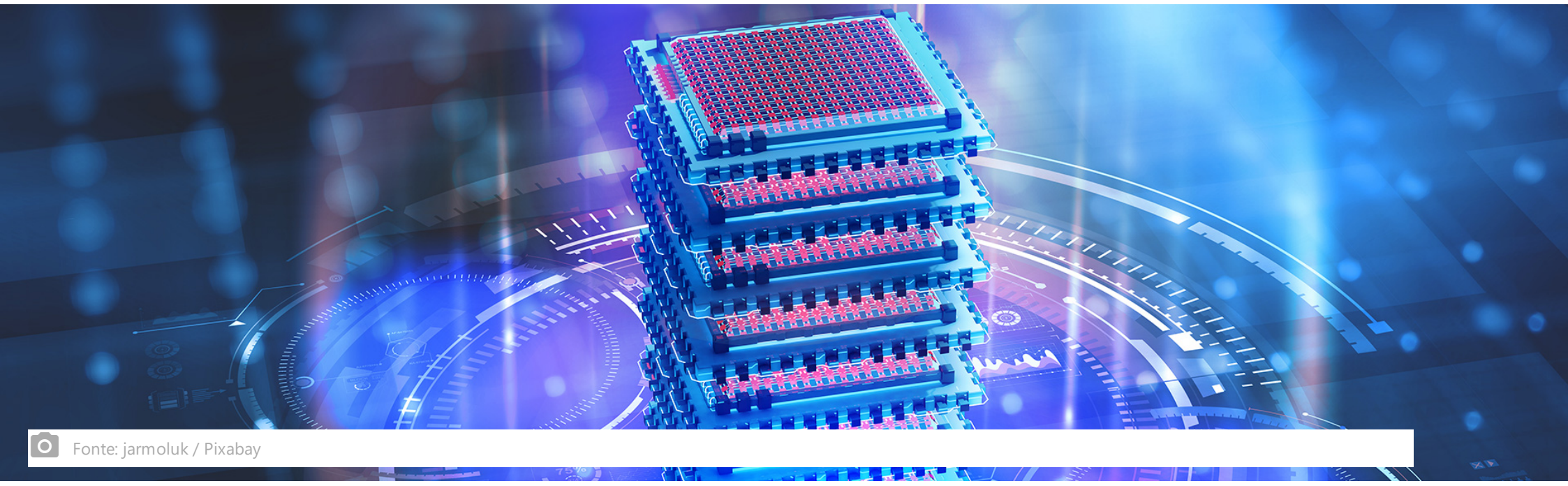
Protegido (protected)

Permite a utilização na classe e nos descendentes, mas não permite acessos externos.

Aqui, quando falamos de descendentes, estamos nos referindo a uma característica importante da orientação a objetos, que é a **herança**, onde utilizamos a palavra reservada **extends**, como pode ser observado a seguir.

```
public class Profissional extends Pessoa{
  public String profissao;
  @Override
  public void exibir() {
    super.exibir();
    // Chama o exibir de Pessoa, imprimindo nome e telefone
    System.out.println("\tTrabalha como "+profissao);
    // Complementa a informação acerca da profissão
  }
}
```

Note que o novo método exibir chama o original através do uso de **super**. Também é necessário colocar a anotação **@Override**, indicando a ocorrência de **polimorfismo**.



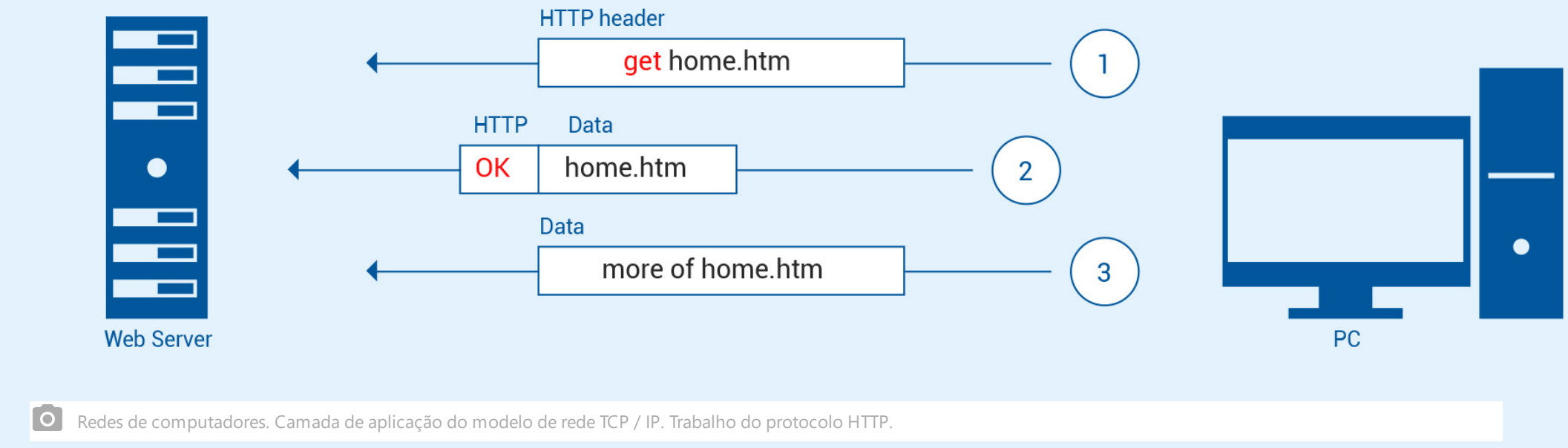
Fonte: jarmoluk / Pixabay

Ambiente Servidor

Aprendemos a criar toda a interface com tecnologias cliente, inclusive a criação de formulários, possibilitando o envio de dados pelos métodos GET ou POST do HTTP.

Mas para quem enviamos os dados?

Os dados são enviados para alguma tecnologia servidora, através de uma requisição HTTP, a qual irá iniciar algum processamento no servidor e este, ao final, retornará uma **resposta**, normalmente em **HTML** ou **XML**.



Redes de computadores. Camada de aplicação do modelo de rede TCP / IP. Trabalho do protocolo HTTP.

No ambiente Java, um objeto da classe **HttpServletRequest** irá:

- **Tratar os dados da requisição HTTP -** normalmente denominado **request**, permitindo efetuar qualquer tipo de processamento com estes dados, como persistência e validações de segurança.
- **Controlar Os dados de retorno para o usuário -** costuma ser denominado response, o qual permitirá escrever o conteúdo HTML ou XML de saída, além de outros elementos, como cookies

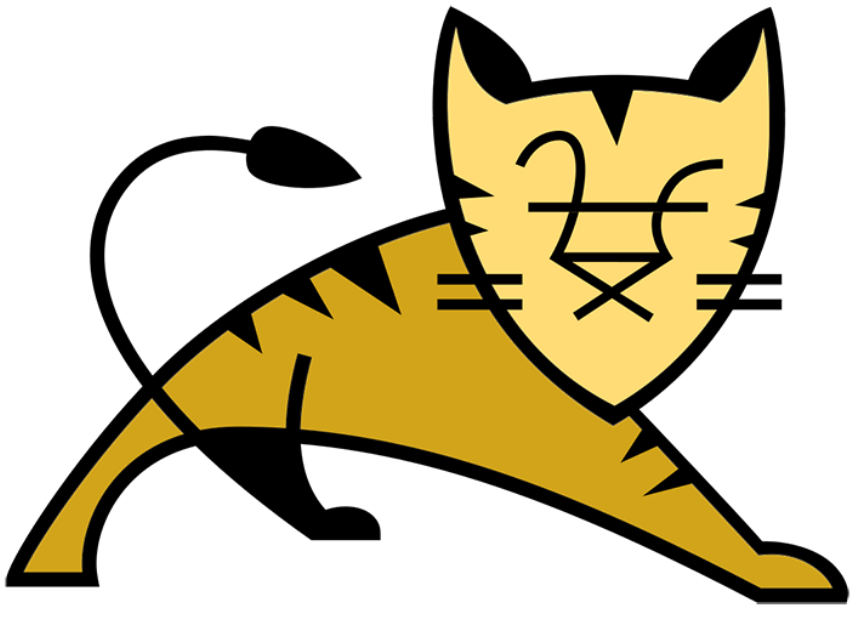
No modelo Web, a cada requisição recebemos uma resposta com uma nova página, como se estivéssemos executando um novo programa, ou seja, todos os objetos antigos deixam de existir, e o estado não pode ser mantido.

A solução para manter estes estados é a utilização de **sessões**, as quais correspondem a objetos alocados ao nível do servidor, fazendo referência a uma dada **conexão**. Enquanto o usuário se mantiver no site, estes dados serão mantidos, sendo eliminados na perda de conexão.

No Java o objeto session, da classe HttpSession, permite a gerência de sessões HTTP, mas é sempre importante lembrar que as sessões consomem memória do servidor, devendo ser utilizadas para fins bem específicos.

Web Server Tomca

O servidor **Tomcat** é um projeto da Apache Software Foundation para a implementação com código aberto de tecnologias como Java Server Pages (JSP), Java Servlet, Java WebSocket e Java Expression Language.



Como qualquer servidor HTTP, o Tomcat também responde às chamadas efetuadas por este protocolo, respondendo com páginas HTML, XML, arquivos binários ou qualquer elemento que possa ser definido através de um tipo MIME, mas a **característica principal dele é que seu container Web fornece suporte às tecnologias de Servlet e JSP**.

Por ser um produto de código aberto, acabou se tornando o padrão para hospedagem de aplicativos Java para Web, podendo atuar como servidor independente ou como módulo de servidores como JBoss e GlassFish.

Apache Software Foundation é a base do ecossistema de software de código aberto moderno, suportando algumas das soluções de software mais usadas e importantes para potencializar a economia da Internet na atualidade.

Mark Driver, diretor de pesquisas da Gartner

Os principais diretórios e arquivos do Tomcat podem ser observados no quadro seguinte:

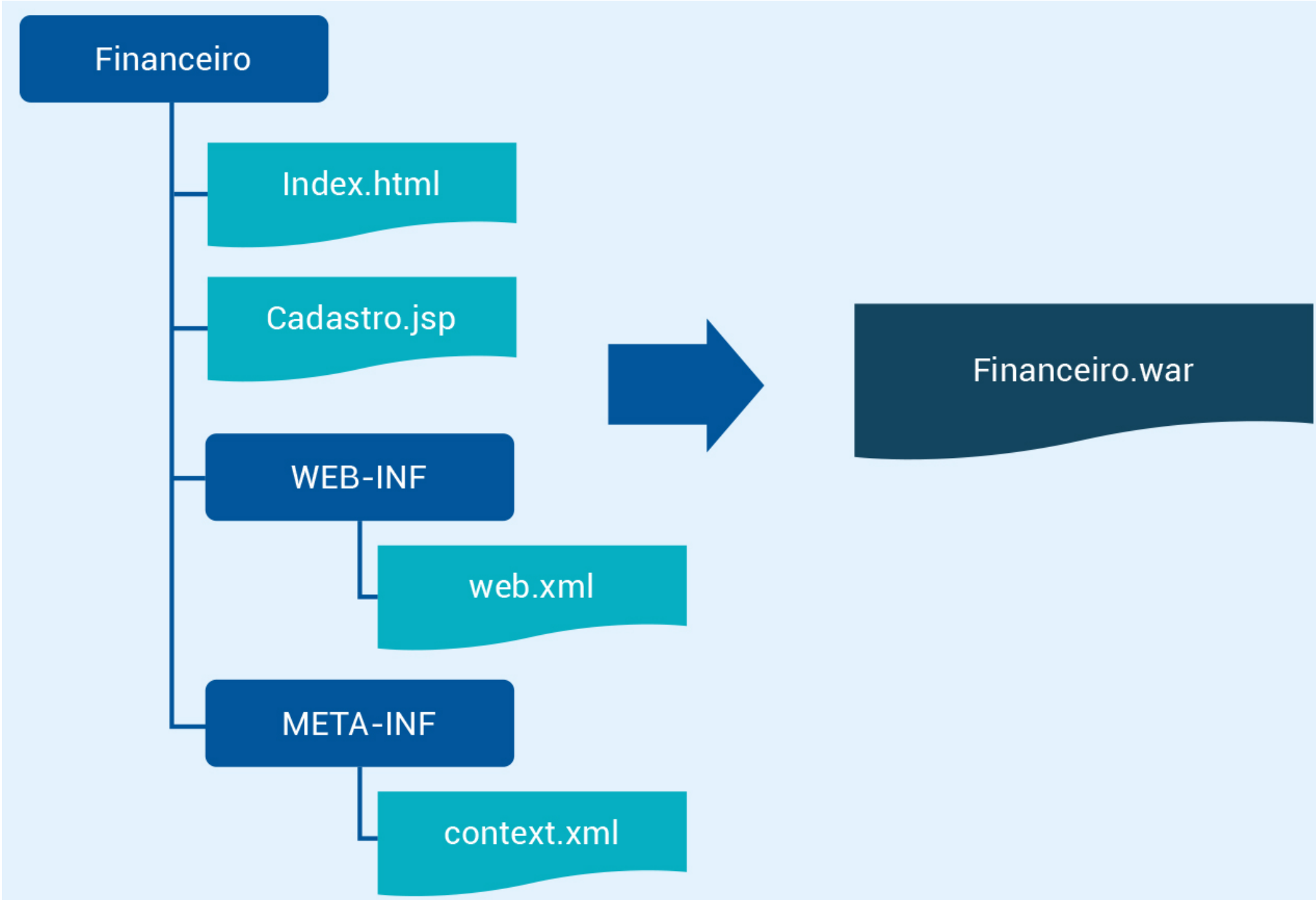
Diretório	Conteúdo
bin	Binários do servidor, incluindo o executável do mesmo.
conf	Arquivos de configuração, como server.xml, que guarda as configurações gerais do Tomcat.
lib	Bibliotecas java de inicialização do servidor no formato jar, as quais também ficam disponíveis para todos os aplicativos do ambiente.
logs	Arquivos de log para identificação de erros na execução do Tomcat.
webapps	Aplicativos Java para Web.

É possível alterar várias características do Tomcat editando o arquivo **server.xml**.

Exemplo

Por exemplo, o servidor Tomcat executa, por padrão, na porta **8080**, mas podemos modificar esta porta procurando e alterando a ocorrência deste valor no arquivo.

Os aplicativos Java Web deverão obedecer a uma estrutura específica, composta por um diretório de base para conteúdo Web, como páginas JSP e HTML, um subdiretório **WEB-INF**, onde fica o arquivo de configuração **web.xml**, um diretório **classes** para o armazenamento das classes Java, e um diretório **lib** com as bibliotecas nos formatos jar e zip, e um subdiretório **META-INF**, onde temos configurações gerais relacionadas aos recursos do servidor, como o arquivo **context.xml**.



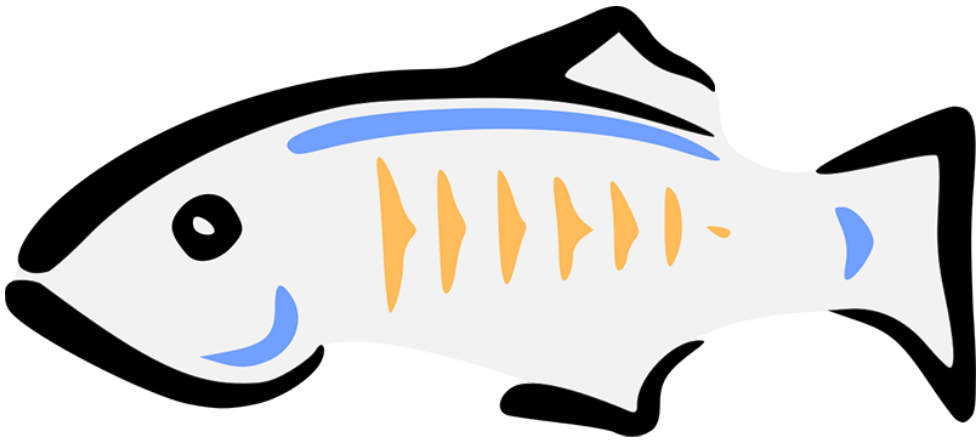
Toda esta estrutura pode ser compactada em um arquivo com extensão **war** (Web Archived), que ao ser copiado para o diretório **webapps**, ou **deploy**, de acordo com a distribuição, é automaticamente expandido, ficando o aplicativo disponível para os usuários.

Essa é uma característica do Tomcat denominada **hot deployment**, fornecendo uma forma muito prática de disponibilizar o aplicativo Web, pois resume o upload de dezenas de arquivos a apenas um.

Application Server GlassFish

Enquanto o Tomcat suporta, de forma nativa, apenas Servlets e JSPs, atuando como um Web Server, o GlassFish vai além, oferecendo suporte às tecnologias Java de Objetos Distribuídos, no caso, os Enterprise Java Beans (EJBs), sendo classificado como Application Server.

Algo interessante a ser mencionado é que o Tomcat é utilizado pelo GlassFish como módulo interno, delegando para ele toda a parte de comunicação HTTP e tratamento de Servlets e JSPs, enquanto os demais elementos da robusta arquitetura do GlassFish tratam das diversas tecnologias do Java Enterprise Edition (JEE).



Com o uso do GlassFish seremos capazes de criar sistemas mais complexos, com uso de EJBs e transações distribuídas, além de obtermos ferramentas para gerenciamento de componentes corporativos, como mensagerias, e ambiente de testes simplificado para Web Services.

Inicialmente o GlassFish era disponibilizado pela Sun, mas após a compra da mesma pela Oracle ele passou a ser denominado Oracle GlassFish Server.

Como o Tomcat não é capaz de prover todas as ferramentas que serão estudadas, adotaremos o GlassFish como ferramenta de trabalho, pois é mais abrangente, dando suporte às tecnologias JEE e cumprindo com todas as funcionalidades de Web Server através do módulo Tomcat interno.

É possível efetuarmos diversas configurações no GlassFish através da interface Web administrativa, ou através do comando **asadmin**, sendo comum o uso de arquivos XML. Por exemplo, uma das opções de configuração é o acréscimo de recursos como **pool de conexões**, elemento que será de grande utilização na criação de sistemas corporativos.

Podemos observar, a seguir, o conteúdo XML necessário para adicionar um pool de conexões.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC
"-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN"
"//glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <jdbc-connection-pool name="SPECjPool" steady-pool-size="100"
  max-pool-size="150" max-wait-time-in-millis="60000"
  pool-resize-quantity="2" idle-timeout-in-seconds="300"
  is-isolation-level-guaranteed="true"
  is-connection-validation-required="false"
  connection-validation-method="auto-commit"
  fail-all-connections="false"
  datasource-classname="oracle.jdbc.pool.OracleDataSource">
    <property name="URL"
    value="jdbc:oracle:thin:@iasperfsol12:1521:specdb"/>
    <property name="User" value="spec"/>
    <property name="Password" value="spec"/>
    <property name="MaxStatements" value="200"/>
    <property name="ImplicitCachingEnabled" value="true"/>
  </jdbc-connection-pool>
  <jdbc-resource enabled="true" pool-name="SPECjPool"
  jndi-name="jdbc/SPECjDB"/>
</resources>
```

Neste exemplo é criado um pool de conexões para um banco Oracle, indicando a URL, usuário e senha, além de características como tempo máximo de espera, utilização de cache, número máximo de comandos SQL paralelos, entre outros.

Também é definido um recurso **JNDI** para viabilizar o acesso do pool a partir de um programa do ambiente Java.

JNDI (Java Naming and Directory Interface) é o serviço de registro e localização de recursos utilizado pelo Java, unificando acesso a componentes nomeados, como pools de conexões, mensagerias, elementos do LDAP, sistemas de arquivos, DNS, entre diversos outros.

Qualquer recurso configurado no servidor GlassFish, para o qual desejamos permitir acesso a partir do Java, deverá estar relacionado a um registro no JNDI. Apenas componentes de uso interno e restrito não apresentarão este registro.

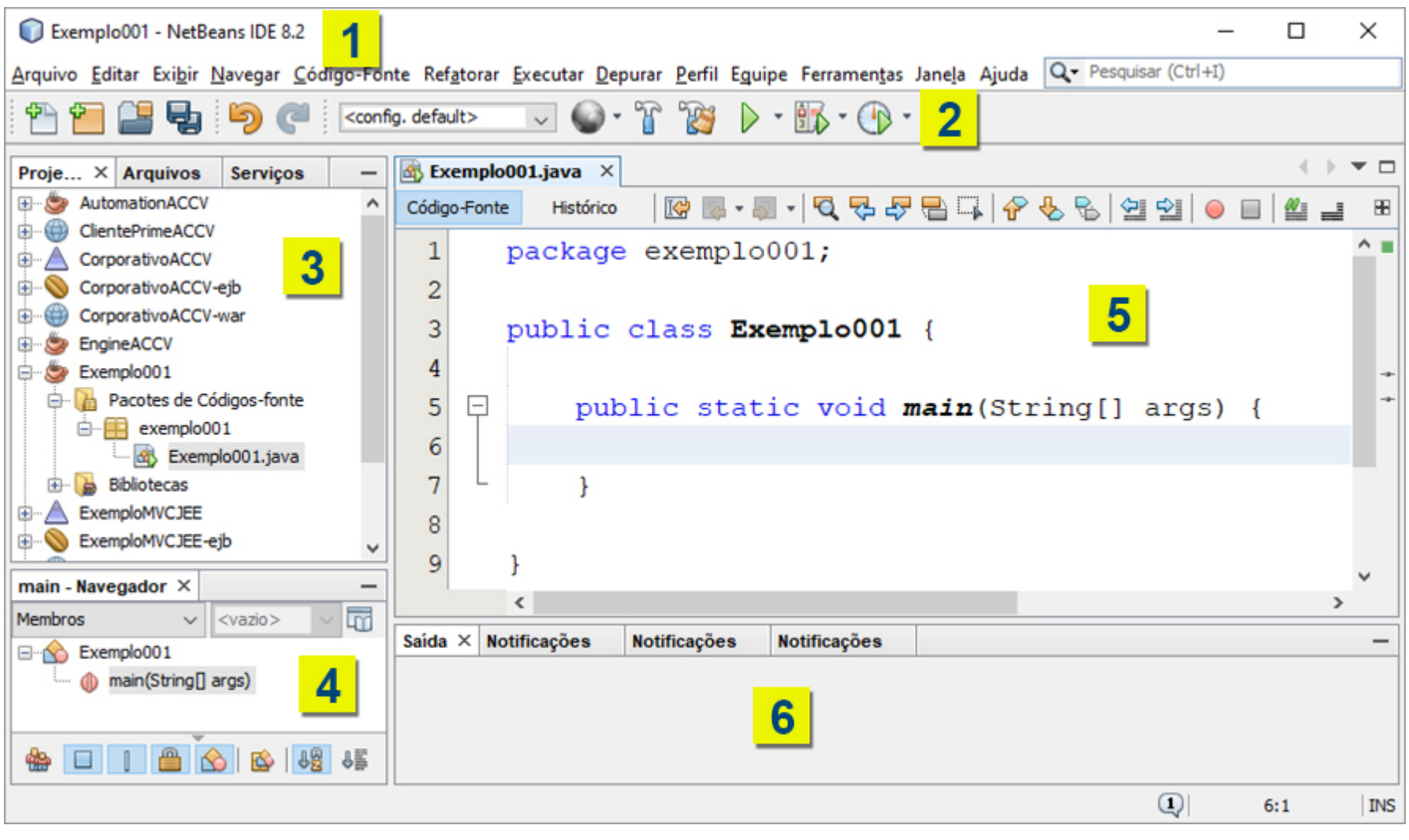
Ambiente de Desenvolvimento

Podemos desenvolver programas em Java com o simples uso de um editor de texto e os programas de compilação e execução oferecidos a partir do **Java Development Kit (JDK)**, o qual pode ser obtido gratuitamente no site da Oracle <https://www.oracle.com> .

Mesmo sendo possível programar em Java com esta abordagem, a produtividade deste método é muito baixa, tornando-se inviável para fins de mercado, o que nos leva a buscar algum ambiente de desenvolvimento integrado (**IDE**); existem várias opções, como Eclipse, NetBeans, JCreator, BlueJ e IntelliJ IDEA.

Nós iremos adotar o NetBeans, por ser mais didático e já trazer uma configuração completa, sem a necessidade de adicionar servidores ou bancos de dados, o que trará maior conforto para todos na aprendizagem da linguagem.

A interface do NetBeans é bastante complexa, e devemos entender seus componentes principais para melhor utilização da ferramenta.



De acordo com a numeração utilizada na figura, temos os seguintes componentes:

1

Menu Principal

Controle global das diversas opções da IDE, ativação e desativação de painéis internos, instalação de plataformas, entre outras diversas funcionalidades. Este é o controle de mais alto nível do NetBeans.

2

Toolbar

Acesso rápido, de forma gráfica, às opções mais utilizadas, como criar arquivos e projetos, salvar arquivos e executar o projeto.

3

Painel de Controle

Aceita várias configurações, mas no padrão normal apresenta uma divisão com a visão lógica dos projetos (**Projetos**), uma com a visão física (**Arquivos**) e outra com acesso a banco de dados e servidores (**Serviços**).

4

Navegador

Permite o acompanhamento das características de qualquer elemento selecionado da IDE, como classes em meio ao código.

5

Editor de Código

Voltado para a edição do código, com diversos auxílios visuais e ferramentais de complementação.

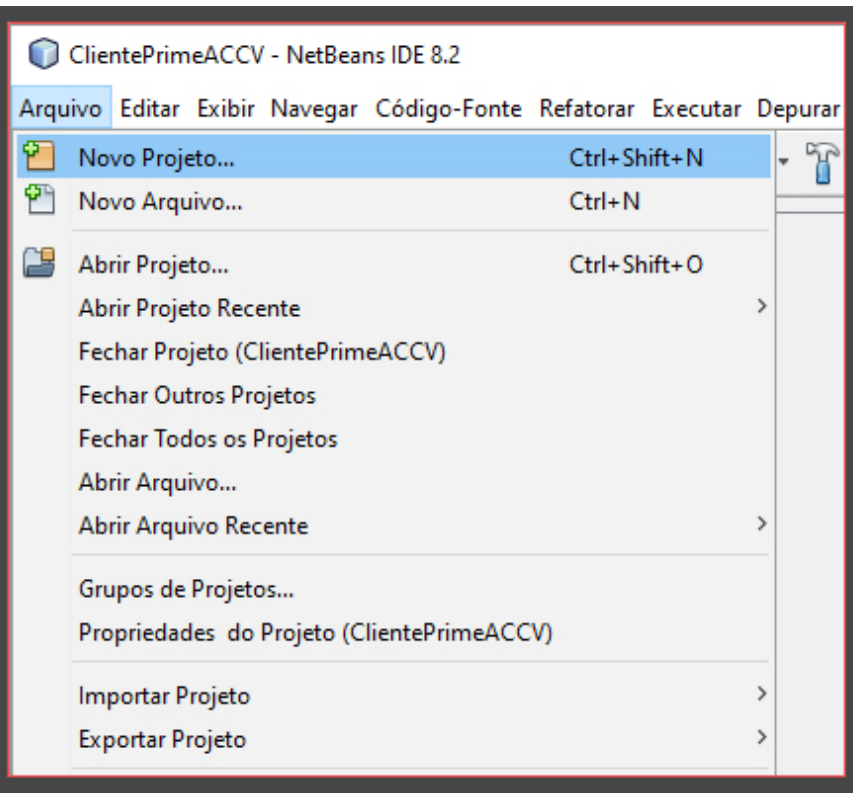
6

Saída

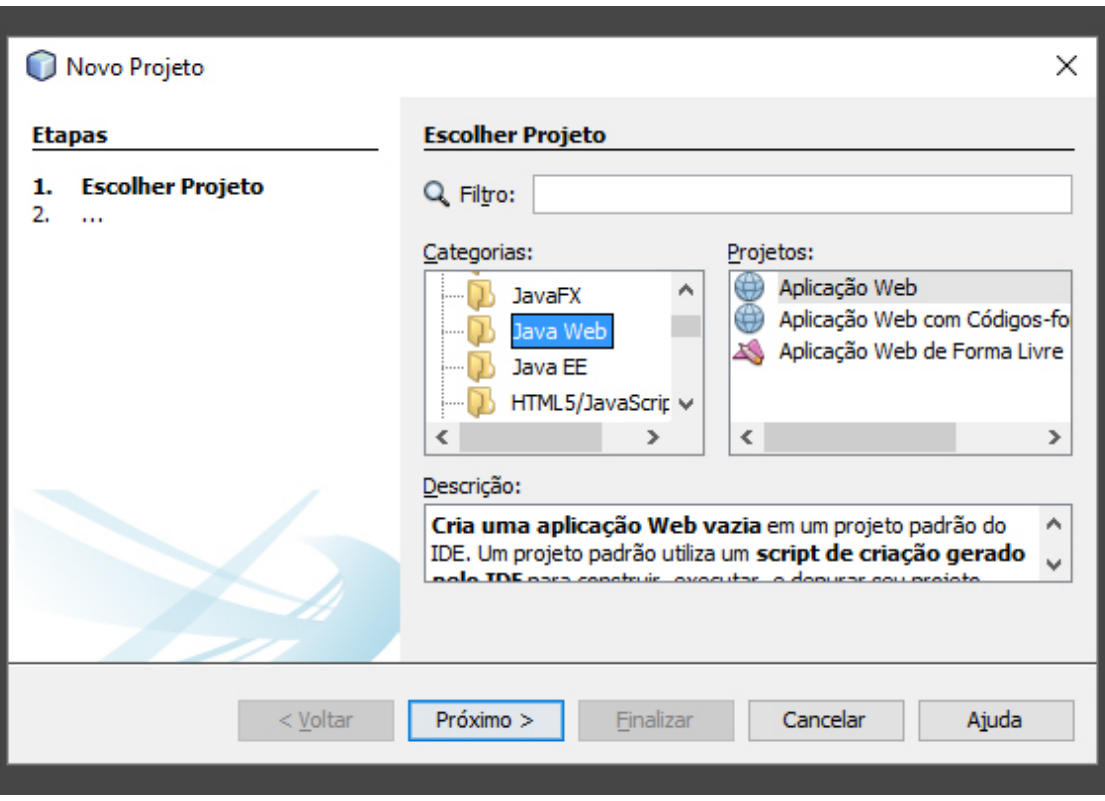
Simula o prompt do sistema, permitindo observar a saída da execução, bem como efetuar entrada de dados via teclado.

O NetBeans pode ser obtido em [netbeans.org/downloads <https://netbeans.org/downloads>](https://netbeans.org/downloads) , devendo ser escolhida a versão completa, pois já traz todas as plataformas de programação configuradas, além dos servidores GlassFish e Tomcat embutidos.

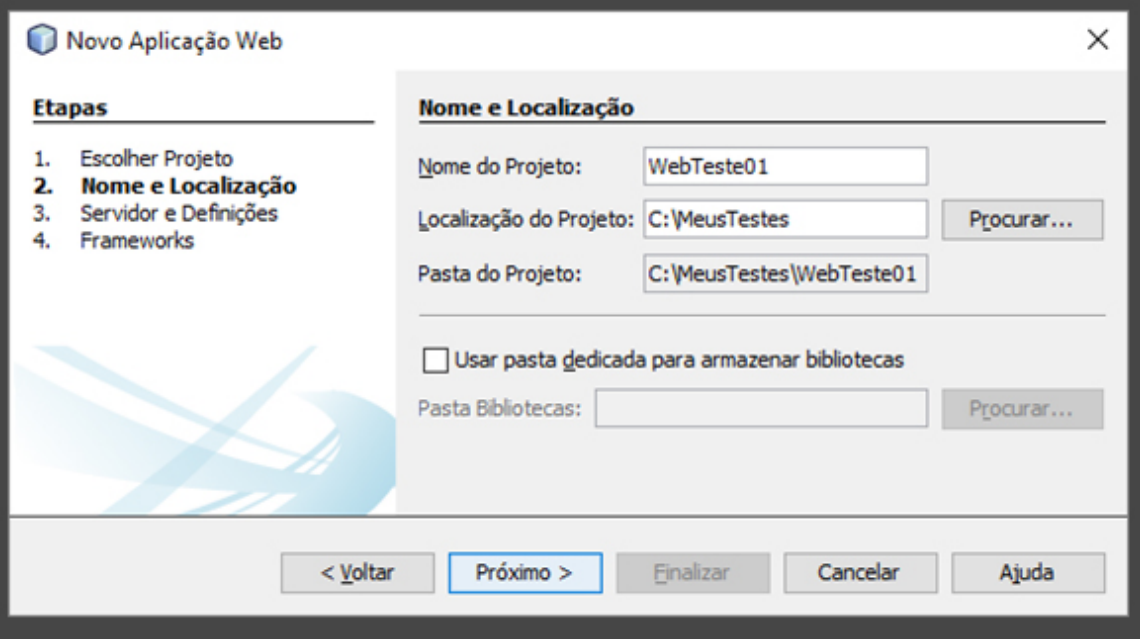
Após instalar o JDK e o NetBeans (verão completa), podemos executar esta IDE e criar o nosso primeiro programa Java para Web, de acordo com a seguinte sequência de passos:



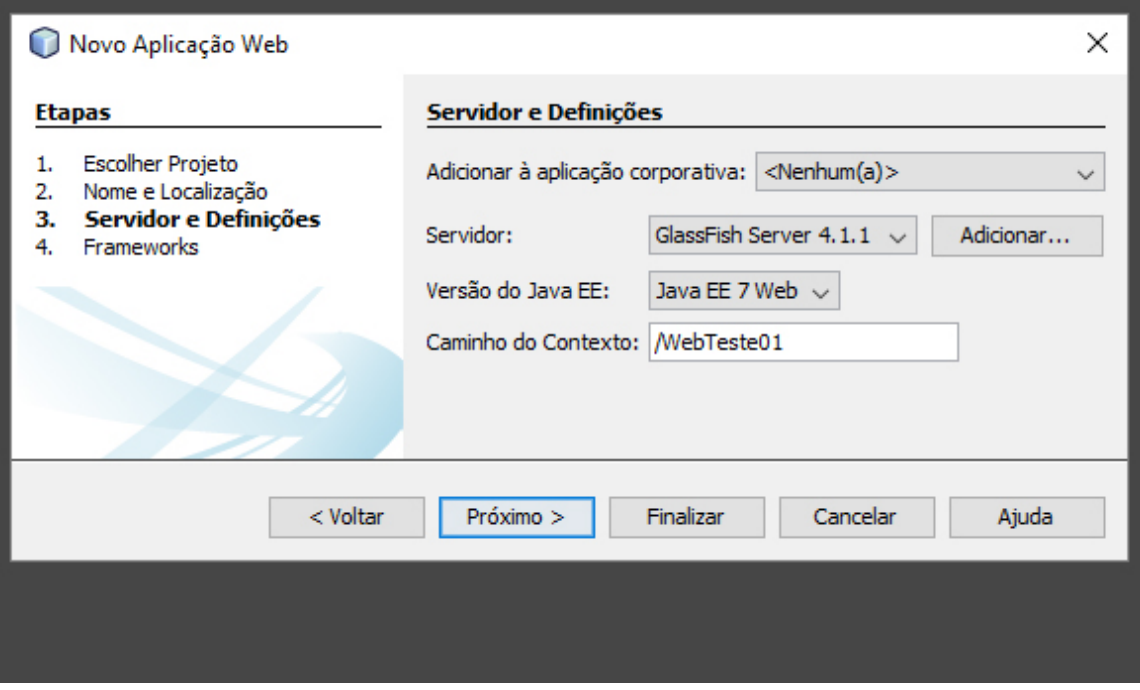
No menu principal do NetBeans escolha a opção **Arquivo..Novo Projeto**, ou **Ctrl+Shift+N**;



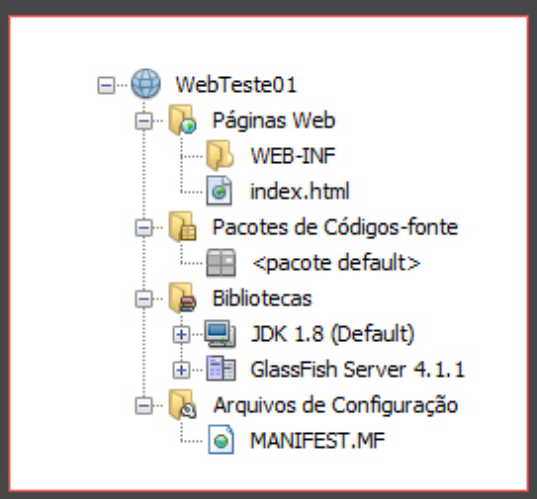
Na janela que se abrirá escolha o tipo de projeto como **Java Web..Aplicação Web** e clique em **Próximo**;



Dê um **nome** para o projeto (WebTeste01) e **diretório** para armazenar seus arquivos (C:\MeusTestes), e clique em **Próximo**;



Escolha o servidor (**GlassFish**) e versão do JEE (**Java EE7**) e clique em **Finalizar**.



Ao término destes passos, o projeto gerado será apresentado no **Painel de Controle**, guia **Projetos**, como pode ser observado na imagem.

Note que o arquivo **web.xml** não é apresentado nesta estrutura, e isso se deve ao fato de termos utilizado o **JEE versão 7**, onde as configurações são efetuadas, em sua grande maioria, com o uso de anotações.

O projeto é dividido em:

- **Páginas Web**, onde se encontram elementos JSP, HTML, CSS e outros;
- **Pacotes de Códigos-Fonte**, onde iremos colocar nossas classes e pacotes Java;
- **Bibliotecas**, para adicionar bibliotecas Java para uso pelo sistema; e
- **Arquivos de Configuração**, incluindo elementos como MANIFEST e web.xml.

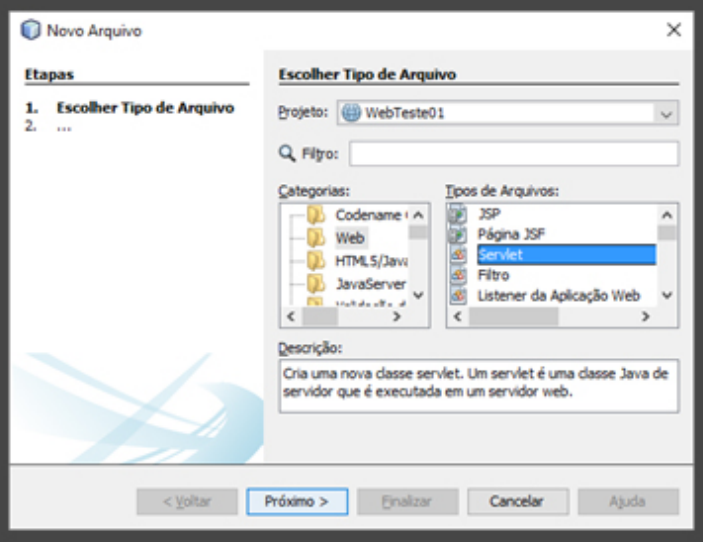
Criado o aplicativo, podemos começar a criar Servlets, JSPs e demais elementos constituintes de nosso sistema, e ao clicar no botão de execução será iniciado o servidor (caso ainda não esteja ativo), gerado o arquivo **war**, copiado este arquivo para o diretório correto do servidor, efetuado o deploy e aberto o navegador no endereço correto para apresentação do **index**.

Servlet

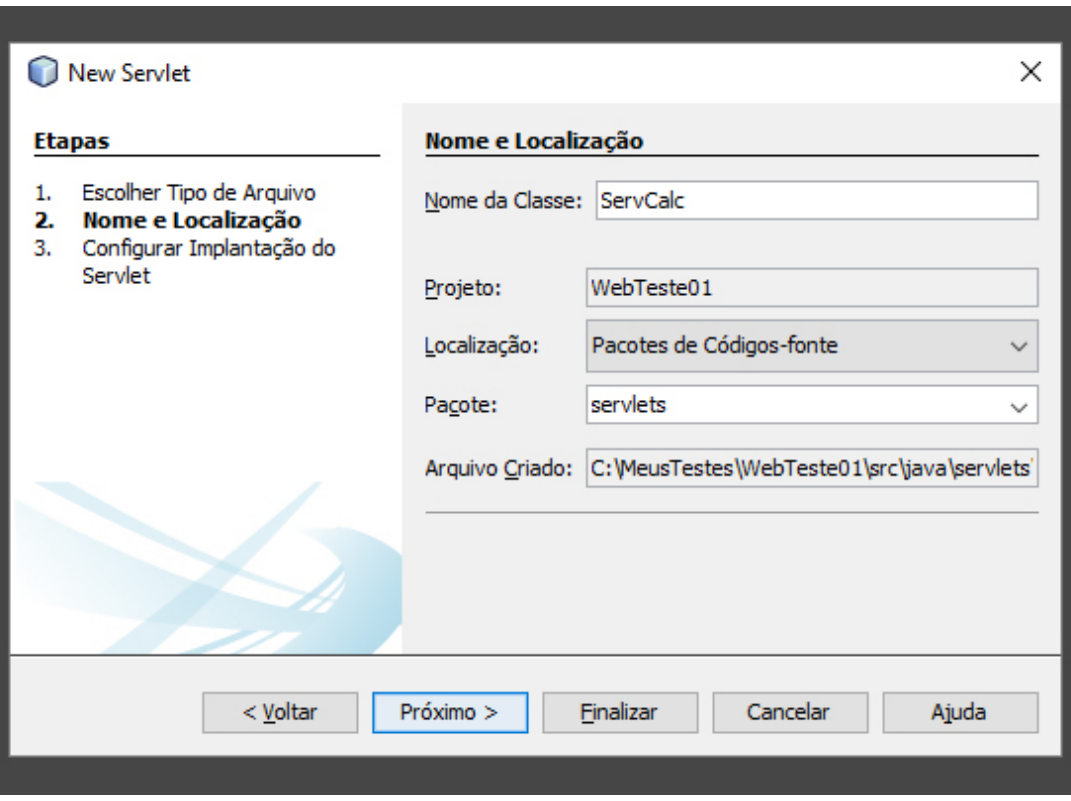
A tecnologia **Servlet** foi criada com o intuito de se tornar uma solução genérica para a criação de aplicativos hospedados em servidores, quaisquer que fossem os protocolos utilizados por eles; mas foi a especialização para o HTTP (**HttpServlet**) que se tornou popular.

Tudo que precisamos fazer é criar um descendente da classe **HttpServlet**, herdando toda a integração com o ambiente já existente, e alterar os métodos **doGet** e **doPost** para personalizar as respostas.

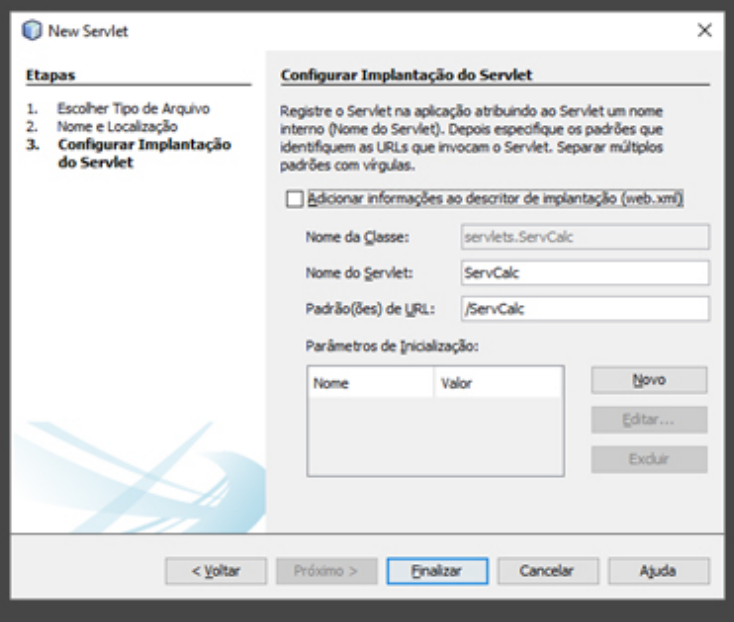
Para adicionar um Servlet ao nosso projeto, devemos escolher o menu **Arquivo..Novo Arquivo**, ou pressionar **CTRL+N**, e seguir os seguintes passos na janela que se abrirá:



Escolha o tipo de arquivo como **Web..Servlet** e clique em **Próximo**;



Dê um **nome** para o Servlet (ServCalc) e para o **pacote** onde será gerado (servlets), e clique em **Próximo**



Observe as configurações de mapeamento do Servlet e clique em **Finalizar**.

O próximo passo é alterar o método processRequest, para que ele corresponda ao processamento necessário em nosso sistema.

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8");
  try (PrintWriter out = response.getWriter()) {
    int a = new Integer(request.getParameter("a")), b = a;
    int fatorial = 1;
    while(a>1)
      fatorial *= a--;
    out.println("<html><body>");
    out.println("O fatorial de "+b+" é "+fatorial);
    out.println("</body></html>");
  }
}
```

Um detalhe no Servlet gerado pelo NetBeans é a presença de código oculto, ou editor-fold, que trata de um trecho de código escondido através de comentários anteriores e posteriores, com a visualização alternada através do clique sobre o sinal de “+” presente na margem esquerda do editor.

O código oculto, neste caso, engloba os métodos **doGet** e **doPost**, responsáveis pela recepção de chamadas HTTP dos tipos GET e POST, respectivamente. No código gerado, ambos os métodos redirecionam para **processRequest**, o que fará com que os dois tipos chamada sejam tratados da mesma forma.

A assinatura de processRequest, assim como doGet e doPost, traz dois parâmetros, o primeiro do tipo HttpServletRequest, encapsulando a requisição HTTP, e o segundo do tipo HttpServletResponse, responsável pela resposta HTTP.

Inicialmente é definido o tipo de saída que será utilizado, através do objeto **response**, e o fluxo da resposta é apontado pelo objeto **out**.

```
response.setContentType("text/html;charset=UTF-8");
try (PrintWriter out = response.getWriter()) {
```

Em seguida nós capturamos o parâmetro enviado pela requisição HTTP, através do objeto request, o qual chega no formato texto, e o transformamos para o tipo inteiro.

```
int a = new Integer(request.getParameter("a")), b = a;
```

Calculamos o fatorial do número no passo seguinte.

```
int fatorial = 1;
while(a>1)
  fatorial *= a--;
```

Finalmente construímos a saída HTML desejada, nesse caso exibindo o fatorial do número fornecido a partir da chamada HTTP.

```
out.println("<html><body>");
out.println("O fatorial de "+b+" é "+fatorial);
out.println("</body></html>");
```

Observe também a presença de uma anotação na definição do Servlet.

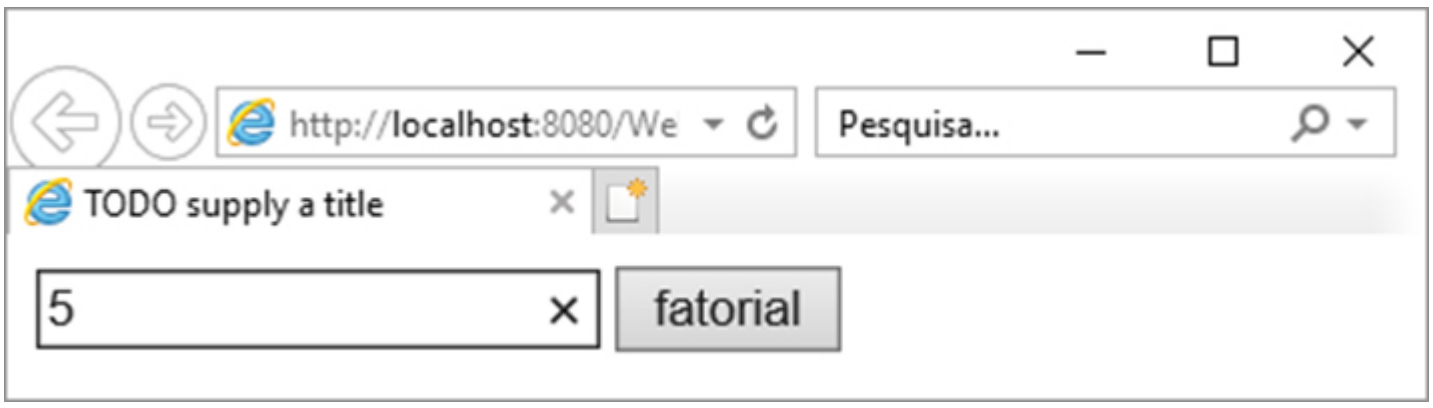
```
@WebServlet(name = "ServCalc", urlPatterns = {"/ServCalc"})
public class ServCalc extends HttpServlet {
```

Esta anotação efetua o mapeamento do Servlet, indicando o nome utilizado por ele e a URL para invocá-lo. Embora tradicionalmente sejam utilizados textos correspondentes ao nome da classe em si, esta não é uma regra, podendo inclusive ser adotadas URLs dinâmicas através de curingas, como “*.jsf”.

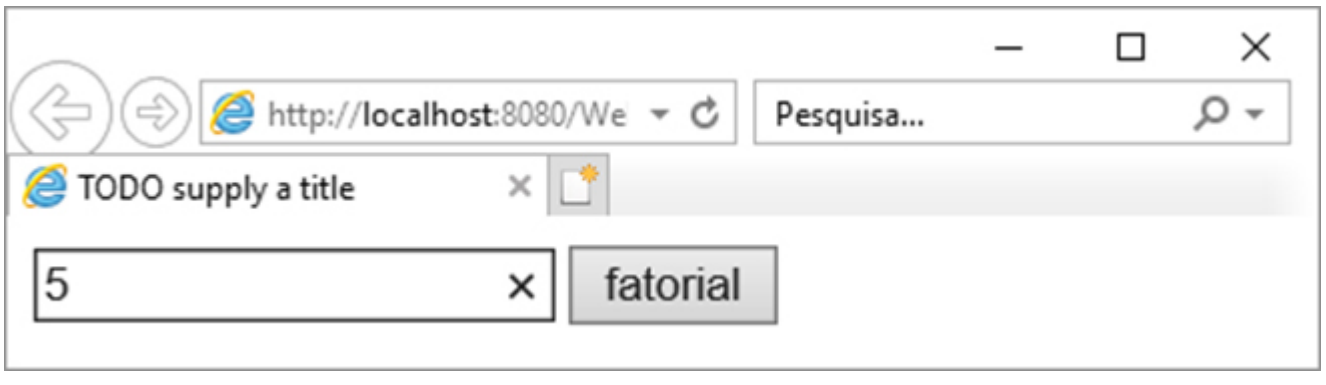
Com o nosso Servlet construído, podemos criar o formulário necessário para efetuar a chamada ao mesmo, e faremos isto com uma pequena inclusão no trecho <body> da página index.

```
  <form method="GET" action="ServCalc">
    <input type="text" name="a"/>
    <input type="submit" value="fatorial"/>
  </form>
 </body>
</html>
```

Finalizando todas estas modificações, precisamos apenas executar o projeto, sendo iniciado o servidor, efetuado o deploy e aberto o navegador na página index.



Preenchendo o número desejado e clicando em somar, nosso Servlet será acionado e a resposta montada, sendo exibida no navegador do cliente.



Java Server Pages

A criação de Servlets seria suficiente para prover as necessidades de todo e qualquer aplicativo Web, porém a construção de páginas através de código direto pode se tornar desconfortável para a maioria dos designers.

Uma solução para isso foi a definição de um novo modelo de programação, no qual códigos Java são escritos dentro do conteúdo HTML ou XML, permitindo a edição visual através de ferramentas como o DreamWeaver da Adobe.

Comentário

A Microsoft já havia implementado este modelo de desenvolvimento através do ActiveX Server Pages (ASP), enquanto no Java foi definida a tecnologia Java Server Pages (JSP).

O processo para a criação de uma página JSP é basicamente o mesmo que o de um Servlet, sendo que devemos escolher arquivo do tipo Web..JSP. O arquivo gerado ficará na seção Páginas Web do projeto ou em um subdiretório da mesma.

Vamos observar a estrutura de uma página JSP simples.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <ul>
    <%
      String[] cores = {"vermelho","verde","azul"};
      for(String x: cores)
      out.println("<li>"+x+"</li>");
    %>
    </ul>
  </body>
</html>
```

A primeira linha deste código é uma **diretiva**, no caso indicando o tipo de conteúdo e a página de acentuação utilizada. Diretivas também são utilizadas para importar bibliotecas, indicar herança, importar taglibs, definir a página de erro, entre diversas outras opções.

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

As linhas seguintes são código HTML padrão, e que não serão executados ao nível do servidor, mas simplesmente irão compor a página de saída.

No meio das tags <ul> temos um **Scriptlet**, que é iniciado com <% e terminado com %>, tratando de código Java que será executado no servidor. Neste código temos um vetor de nomes de cores, os quais serão impressos no conteúdo da página através do objeto out, implícito nas páginas JSP.

```
<%
String[] cores = {"vermelho","verde","azul"};
for(String x: cores)
    out.println("<li>"+x+"</li>");
%>
```

Para executar diretamente o JSP, basta clicar com o botão direito sobre ele na guia de **Projetos** e mandar executar arquivo. Podemos ver a página gerada a seguir.

Os objetos **request** e **response** também são implícitos para as páginas JSP, podendo ser utilizados da mesma forma que o foram nos Servlets.

Atenção

Inclusive, algo que devemos lembrar é que toda página JSP é convertida em Servlet pelo container Web no primeiro acesso. Logo, o que muda é basicamente a forma de programar, e não a funcionalidade original.

Atividade

1. Qual tipo de classe permite manter valores entre chamadas sucessivas no ambiente Java Web?

a) HttpServletRequest
b) HttpServletResponse
c) HttpServlet
d) HttpListener
e) HttpSession

2. Implemente o método processRequest, de um Servlet criado a partir do NetBeans para receber dois números via HTTP e retornar a soma entre eles.

3. Crie uma página JSP para apresentar os números ímpares entre 1 e 100 no formato de lista HTML.

Referências

CASSATI, J. P. **Programação servidor em sistemas web**. Rio de Janeiro: Estácio, 2016.

DEITEL, P; DEITEL, H. **Ajax, rich internet applications e desenvolvimento web para programadores**. São Paulo: Pearson Education, 2009.

_____. **Java, como programar**. 8. Ed. São Paulo: Pearson, 2010.

SANTOS, F. **Tecnologias para internet II**. Vol. 1. Rio de Janeiro: Estácio, 2017.

Próxima aula

- Sintaxe SQL para bancos de dados relacionais
- Conceito de Middleware;
- Utilização do JDBC e padrão DAO para acesso ao banco de dados.

Explore mais

Não deixe de visitar as seguintes páginas:

- Tutorial de Java da IBM (Utiliza Eclipse) <https://www.ibm.com/developerworks/br/java/tutorials/j-introtojava1/index.html>
- Tutorial de criação de projeto do NetBeans <https://netbeans.org/kb/docs/java/quickstart_pt_BR.html>
- Instalação do NetBeans e JDK" <https://www.youtube.com/watch?v=GNUGNfIIGhA>
- Configuração do Tomcat no Eclipse <https://www.youtube.com/watch?v=9Z4OKoh-Omw>
- Manual do GlassFish <https://javaee.github.io/glassfish/doc/5.0/reference-manual.pdf>
- Tutorial de Servlet e JSP <https://www.journaldev.com/2114/servlet-jsp-tutorial>
- Programação em Java <//www.faeterj-rio.edu.br/downloads/bbv/0031.pdf>