

# Laboratório de Programação de Web Sites Dinâmicos

Apresentação da Disciplina

Tassio Sirqueira – 2019/02

# Apresentação da Disciplina | Sobre vocês...

- Quais linguagens você conhece?
  - Como está seu conhecimento de Java?
- Já fez a disciplina de Orientação à Objetos?
- Já sabe montar páginas com HTML5 / CSS?
- Já fez algum script utilizando JavaScript?
  - JQuery?
  - Ajax?
- Já fez um projeto com Java e banco de dados?



# | Apresentação da Disciplina | Sobre mim...

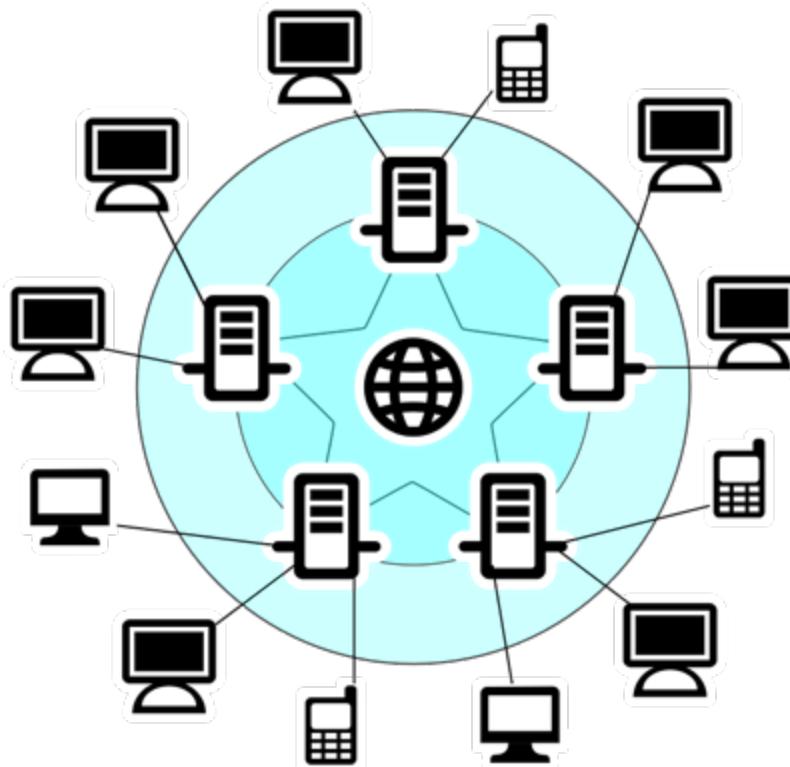
- ❑ Bacharel em Sistemas de Informação pelo CES/JF.
- ❑ Mestre em Ciência da Computação pela UFJF.
- ❑ Doutorando em Informática pela PUC-Rio.
- ❑ Atualmente Analista na MRS, Professor no CES/JF e avaliador do INEP/MEC.



Site: <https://tassio.eti.br/>  
E-mail: tassio@tassio.eti.br

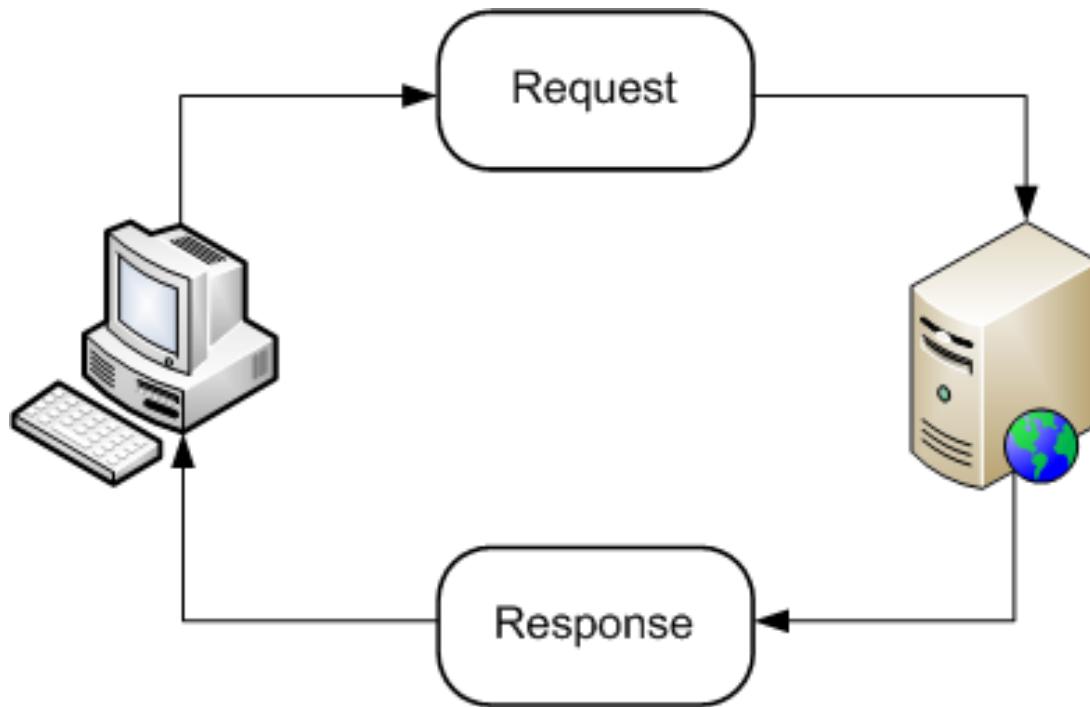
# | Apresentação da Disciplina | Introdução

- ❑ O que é a internet?



# | Apresentação da Disciplina | Introdução

- ❑ Requisitar e Receber recursos



# Apresentação da Disciplina | Introdução

## □ Páginas Web

WIKIPEDIA  
The Free Encyclopedia

Main Page | Recent changes | Protected page | History | Special pages | Go | 209.237.238.158 | Log in | Help | Search

Printable version | Other languages: German | Esperanto | Spanish | French | Dutch | Polish | Portuguese

### Main Page

From Wikipedia, the free encyclopedia.

Welcome to Wikipedia, a collaborative project to produce a complete, freely editable encyclopedia. You are already working on 48152 articles, with more being added and improved now, without even having to log in. You can copyedit, expand an article, or add background information about the project, and the help page for information on how to do this.

The content of Wikipedia is covered by the [GNU Free Documentation License](#) for the details and [open content](#) and [free content](#) for the license.

Protected page

Main Page | Recent changes | Random page | Watch list | Current events | Talk page | History | What links here | Watch links | Bug reports | Special pages

Background on current events

Current events - Oktoberfest - Israeli-Palestinian conflict - West Bank - Robert L. Forward - Apollo 12

Philosophy, Mathematics, and Natural Science

Astronomy and astrophysics - Biology - Chemistry - Earth science

Social Sciences

Anthropology - Archaeology - Economics - Geography - History

Google search results for "opra":

Searched About Google pages for opra.

**Opra to the Pensions Regulator redirect page**

On 6 April 2005 the Pensions Regulator took over from Opra (the Occupational Pensions Regulatory Authority). The Pensions Regulator is the new regulatory ...  
[www.thepensionsregulator.gov.uk/tprredirect.htm](http://www.thepensionsregulator.gov.uk/tprredirect.htm) - [Similar pages](#)

**The Pensions Regulator**

Skip to: Content. Links: Accessibility. Cymraeg. Contact us. Site map. Opra archive. Advanced search. Sections: About Us. Stakeholder pensions. ...  
[www.thepensionsregulator.gov.uk/](http://www.thepensionsregulator.gov.uk/) - [Similar pages](#)  
[ More results from [www.thepensionsregulator.gov.uk/](http://www.thepensionsregulator.gov.uk/) ]

**Open Public Records Act**

OPRA Central OPRA|New Jersey Open Public Records Act ... Here users can view information regarding OPRA, contact the GRC and register a complaint online. ...  
[www.state.nj.us/opra/](http://www.state.nj.us/opra/) - [Similar pages](#)

# Apresentação da Disciplina | Introdução

## □ Aplicações Web

The image shows two side-by-side screenshots of a web-based email application interface, likely Gmail, demonstrating its features.

**Left Screenshot (Inbox View):**

- Header:** Mail, Calendar, Documents, Photos, Groups, Web, more ▾
- Search Bar:** Search Mail, Search the Web, Show search options, Create a filter
- Title Bar:** BusinessWeek Online -- Dell Talking Again After Audit - 2 hours ago
- Toolbar:** Archive, Report Spam, Delete, More actions..., Refresh
- Message List:** Shows several messages from Luis Verbera, Ticketmaster, Social Edge, support, sales, service@paypal.com, service@paypal.com, service@paypal.com, Live Nation, Live Nation, Thompson Alves, I Love My Local Blood Ba., Luis, me (2), Rock the Bells Festival, craigslist.org, APE Concerts, TokBox (2), Reginald Elio, TokBox, CrunchForums, TokBox, TokBox, TokBox.
- Left Sidebar:** Compose Mail, Inbox (31), Stared (1), Chats, Sent Mail, Drafts, All Mail, Spam (113), Trash, Contacts, Labels (e.g., admeyer, at&t, buzzyah, craigslist, freshwoldsalad, frothbuzz, hostican, techonunch), Invite a friend, Give Gmail to:, Send invite, preview invite.

**Right Screenshot (Spam Filter View):**

- Header:** +Barbara Gmail Calendar Documents Photos Sites Web More ▾
- Search Bar:** in:spam, Search Mail, Search the Web, Show search options, Create a filter
- Title Bar:** Spam Swiss Pie - Bake 45-55 minutes or until eggs are set
- Toolbar:** Delete forever, Not spam, Mark as read, Move to, Labels, More, C
- Text:** Delete all spam messages now (messages that have been in Spam more than 30 days will be automatically deleted)
- Message List:** Shows many spam messages from various senders, such as Loan Department, davielv9, mao, Background Check Alert, HANNAH Holiday, MACKENZIE Winter, LEAH Aldridge, AARP Membership, AOL Administration Cente., like, mkt, robert.rethdd, Online Store, Cheryl Perez, ADDISON Nelson, Joshua Hulkmen (STUDEN.), ASHLEY Galbraith, Rex Todd, Susan's Business Blog, NORA Brickman.
- Left Sidebar:** Compose mail, Mail (in:spam), Contacts, Tasks, Inbox (31), Stared (1), Important, Sent Mail, Drafts (2), Spam (139), Trash, [imap]/Sent, [imap]/Trash (2), bwriter (2), Daily SF, Droid, IPG (14), 14 more ▾, Chat.

# Apresentação da Disciplina | Introdução

Aplicações Web Modernas

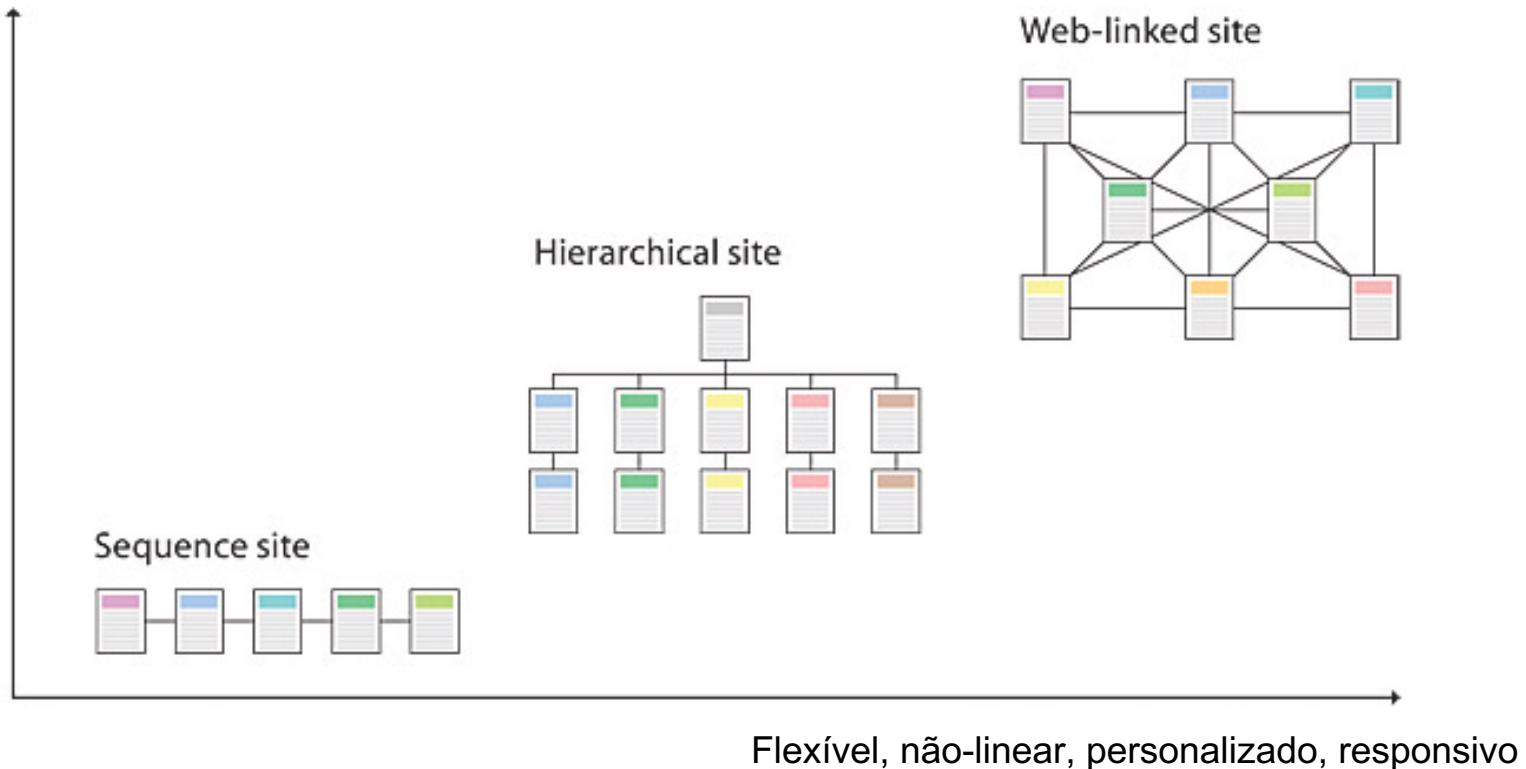
# Apresentação da Disciplina | Introdução

- ❑ Hoje as Aplicações Web São Muito Complexas



# Apresentação da Disciplina | Introdução

Complexidade,  
robustez,  
segurança



# Apresentação da Disciplina | Introdução

- ❑ Desenvolver softwares web que atendam a todos esses requisitos é uma tarefa desafiadora.
  - ❑ Muitos Requisitos funcionais.
  - ❑ Demanda por interfaces atraentes
  - ❑ Muitos tipos de interação
  - ❑ Múltiplos tamanhos de tela
- ❑ Além das regras de negócio, experiência do usuário e interface das aplicações web, existem uma série de preocupações relacionadas a **segurança, arquitetura e infraestrutura** que precisam ser tratadas.

# Apresentação da Disciplina | Introdução

- ❑ Preocupações ao se desenvolver uma aplicação web moderna:
  - ❑ Modelo persistência em banco de dados;
  - ❑ Integridade de transações;
  - ❑ Integração com outras aplicações / web services;
  - ❑ Gerenciamento de múltiplos usuários ao mesmo tempo;
  - ❑ Gerenciamento de conexões HTTP;
  - ❑ Cache de objetos;
  - ❑ Gerenciamento da sessão web;
  - ❑ Autenticação e Autorização;
  - ❑ Balanceamento de carga
  - ❑ ....

# | Apresentação da Disciplina | Introdução

- ❑ Esse esforço dedicado à infraestrutura diminui a capacidade de entrega.
- ❑ Prejudica a qualidade.
  - ❑ Especialistas em programação concorrente? Segurança? Sistemas distribuídos?



# | Apresentação da Disciplina | Introdução

- ❑ Para ajudar a resolver esse problema, a Sun propôs uma série de especificações.
- ❑ Pudessem ser implementadas e oferecer toda essa infraestrutura já pronta.
- ❑ Diferentes fornecedores e empresas poderiam implementar essas especificações e os desenvolvedores não precisariam se preocupar.

# Apresentação da Disciplina | Introdução

- ❑ Esse conjunto de especificações formam o **Java EE (Java Enterprise Edition)**.
  - ❑ Série de especificações bem detalhadas, dando uma receita de como deve ser implementado cada um desses serviços de infraestrutura.
- ❑ Vantagens
  - ❑ Os desenvolvedores podem focar apenas em suas regras de negócio.
  - ❑ Diferentes empresas podem oferecer software para a infraestrutura a partir das especificações definidas.
  - ❑ Os desenvolvedores não ficam presos a um fornecedor específico, podendo trocá-lo sem necessidade de reescrever todo o código.

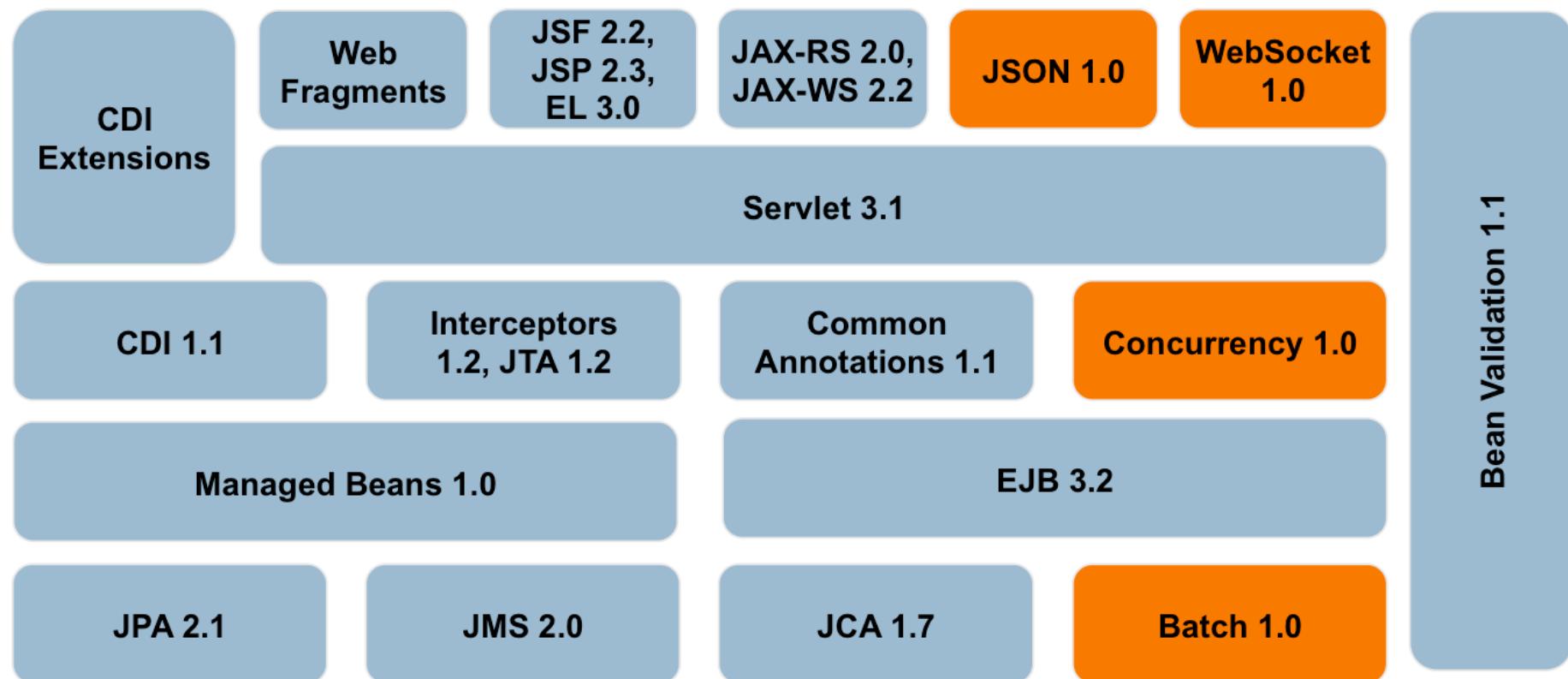


# Apresentação da Disciplina | Java EE

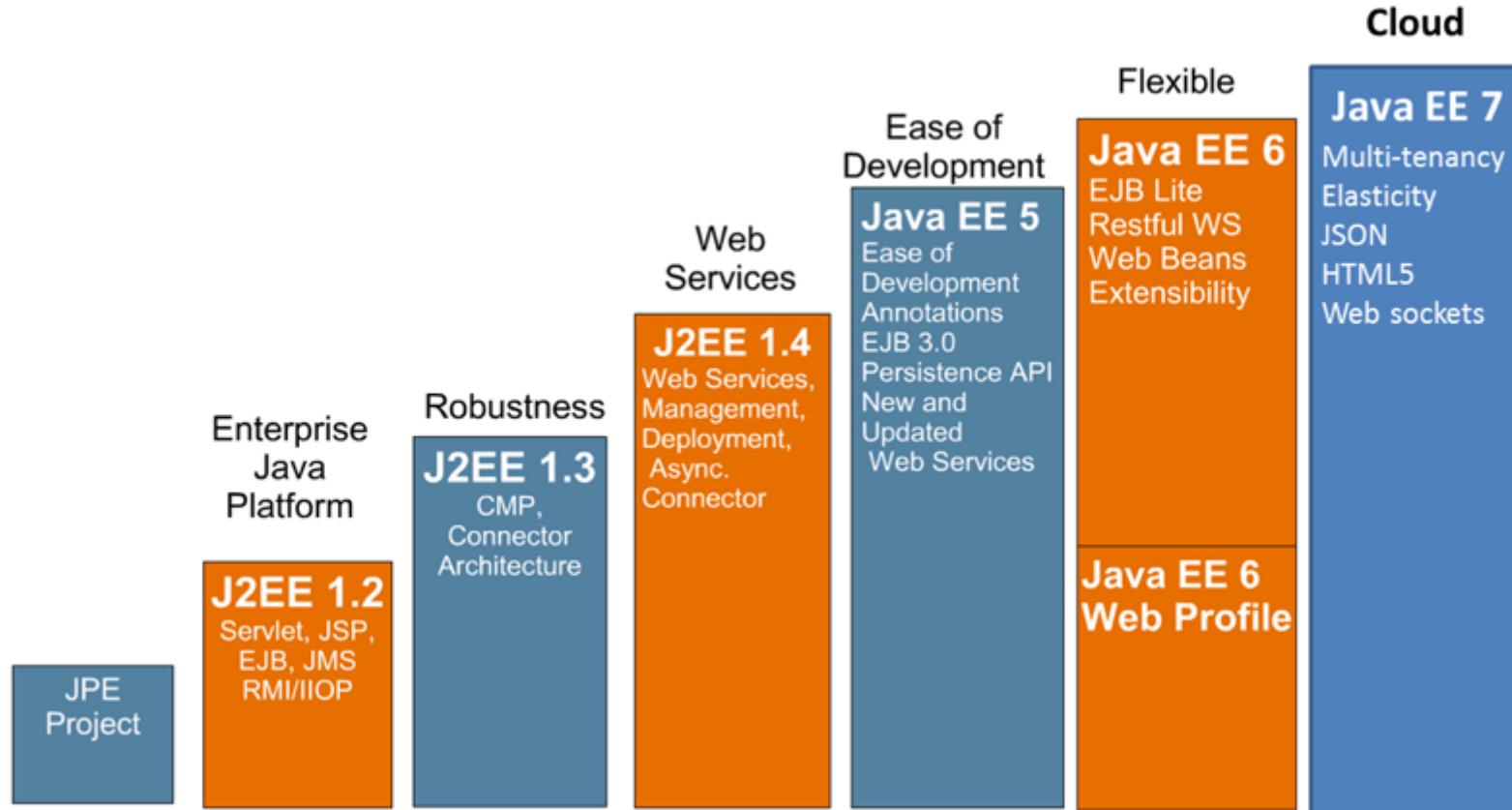
## Principais especificações do Java EE

- JavaServer Pages (**JSP**), Java Servlets, Java Server Faces (**JSF**)
  - Trabalhar para a Web.
- Enterprise Javabeans Components (**EJB**) e Java Persistence API (**JPA**)
  - Objetos distribuídos, clusters, acesso remoto a objetos, etc.
- Java API for XML Web Services (**JAX-WS**), Java API for XML Binding (**JAX-B**)
  - Trabalhar com arquivos xml e webservices.
- Java Autentication and Authorization Service (**JAAS**)
  - API padrão do Java para segurança.
- Java Transaction API (**JTA**)
  - Controle de transação no contêiner.
- Java Message Service (**JMS**) , Java Naming and Directory Interface (**JNDI**)  
(espaço de nomes e objetos), Java Management Extensions (**JMX**)....

# Apresentação da Disciplina | Java EE



# Apresentação da Disciplina | Java EE



## Java 8

- ❑ Melhorias em Interfaces.
- ❑ Expressões Lambdas.
- ❑ Generics.
- ❑ Nova API de Data e Hora.
- ❑ Outras APIs.



# | Apresentação da Disciplina | Java EE

## Java EE Ecosystem



# Apresentação da Disciplina | Java EE

- ❑ Como eu faço download do Java EE?
  - ❑ Você não ia querer, são apenas páginas e mais páginas em PDF.
- ❑ Servidores de Aplicação
  - ❑ São os softwares que implementam essa especificação e oferecem todos os recursos do Java EE prontos para serem utilizados.
  - ❑ A própria Oracle desenvolve uma dessas implementações, o **Glassfish** que é open source e gratuito.



# | Apresentação da Disciplina | Java EE

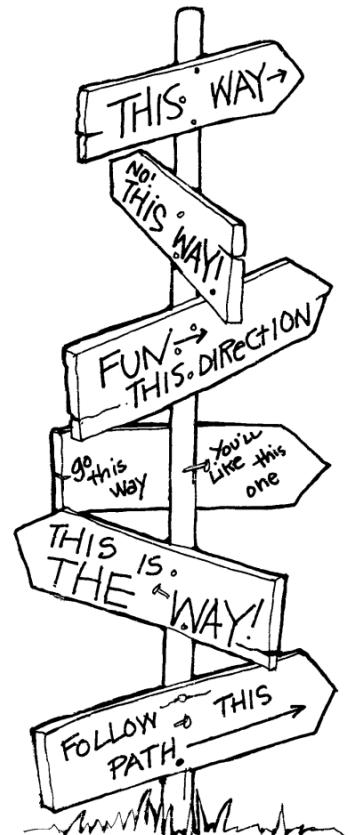
Existem outros Servidores de Aplicação

- GlassFish Server, gratuito
- JBoss WildFly
- Oracle/BEA, Oracle WebLogic Server
- IBM, IBM WebSphere Application Server



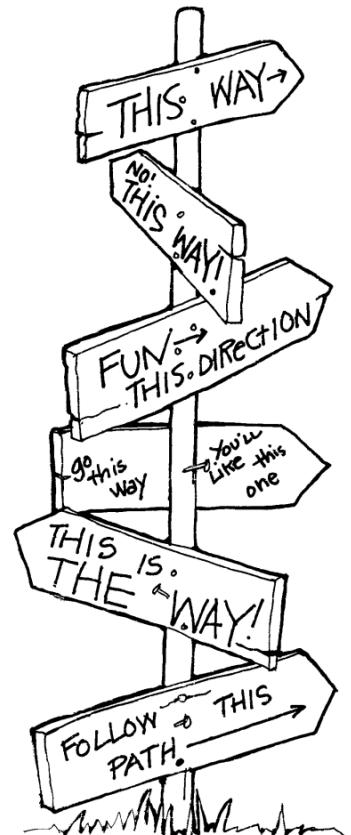
# | Apresentação da Disciplina | Ementa

- ❑ Revisão de tecnologias web
- ❑ Comunicação na web: HTTP
- ❑ Desenvolver Web Sites Dinâmicos
  - ❑ Servlets, JSP, JSF...
- ❑ Aprender padrões de projeto e arquitetura de aplicações web corporativas.
- ❑ Estudo de caso



# | Apresentação da Disciplina | Avaliação

- Avaliação
  - Peso: 2
- Avaliação
  - Peso: 3
- Trabalho
  - Peso: 3
- Atividades em Sala
  - A média das atividades entregues em dia vale 1 ponto extra.





# Laboratório de Programação de Web Sites Dinâmicos

HTTP

Tassio Sirqueira – 2019/02

# HTTP | Principais Conceitos

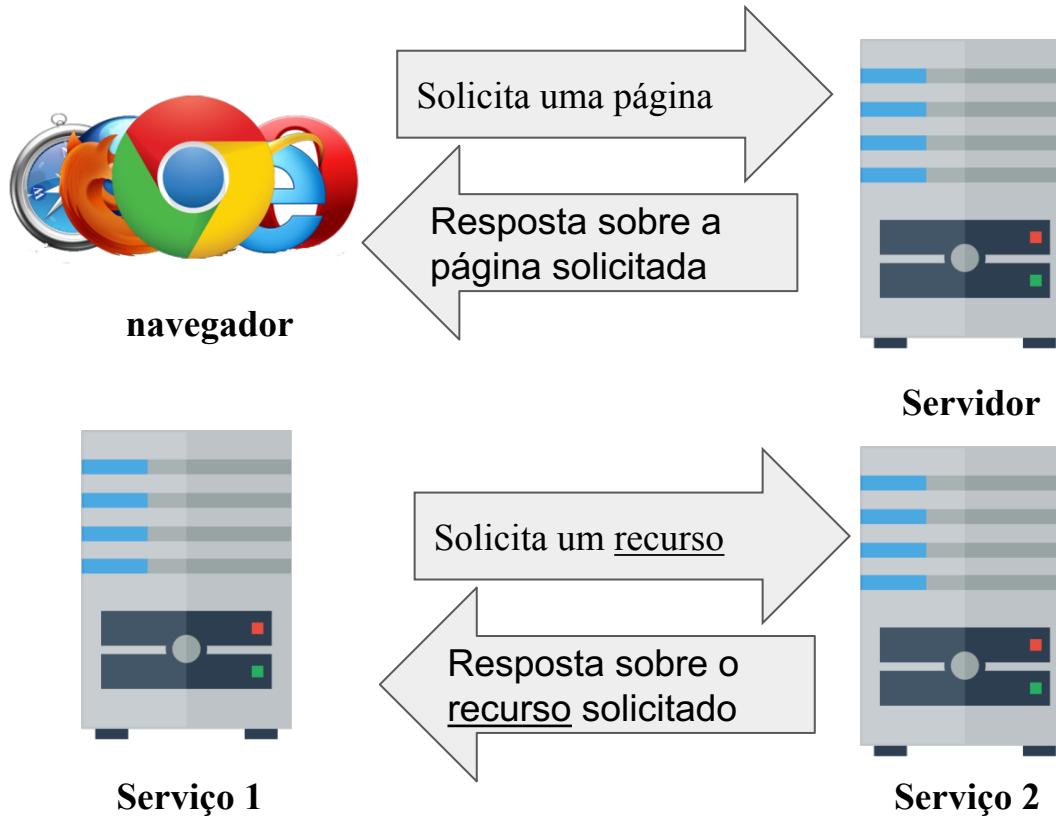
## HTTP

7	Aplicação	HTTP, FTP, DNS, DHCP, ...
6	Apresentação	EBCDIC, NDR, ...
5	Sessão	RCP, SSH, SCP, NetBios...
4	Transporte	TCP, UDP, ...
3	Rede	IP, IPX, ICMP, ARP, RARP, ...
2	Ligação de Dados	Ethernet, FDDI, Frame relay ...
1	Física	Modem, camada física ethernet, ...

A Internet moderna é inteiramente baseada no protocolo Hypertext Transfer Protocol (**HTTP**).

- ❑ Protocolo de comunicação da camada de aplicação.
- ❑ Não possui estado, portanto não tem ‘memória’ das requisições anteriores.
- ❑ Especificado nas RFCs (7230-7237).

# HTTP | Exemplo de comunicação HTTP



# | HTTP | HTTP

- ❑ O HTTP se resume a requisições e respostas.
  - ❑ Assume muito pouco a respeito dos detalhes de implementação.
  - ❑ A versão mais utilizada é a HTTP 1.1, embora já exista a versão HTTP 2.0.



# HTTP | Recursos

- ❑ Na *web* qualquer endereço que pode ser acessado é um **recurso**.
- ❑ Representamos esses endereços por meio das **URLs** (Uniform Resource Locators)
  - ❑ `http://www.google.com`
  - ❑ `http://globoesporte.globo.com/`
- ❑ Uma URL pode possuir várias partes.

<code>http://</code>	<code>www.domain.com:</code>	<code>1234/</code>	<code>path/to/resource?</code>	<code>a=b&amp;x=y</code>
Protocolo	Servidor (IP, DNS..)	Porta	Caminho para o recurso	Parâmetros de Busca

# HTTP | Recursos

❑ `https://www.google.com.br/search?q=o+que+é+http`

Protocolo

Servidor  
(IP, DNS..)

Caminho  
para o  
recurso

Parâmetros de Busca

A porta é opcional, se omitida é  
utilizada a porta padrão do TCP  
80.

❑ `http://localhost:8080/meuprojecto`

Protocolo

Servidor  
(IP, DNS..)

Caminho para o  
recurso

O uso de parâmetros de busca é  
opcional.

# HTTP | Verbos

- ❑ Uma URL indica um recurso disponível no Servidor com o qual queremos nos comunicar.
- ❑ No entanto, a ação que deve ser realizada é definida pelo **Verbo**:
  - ❑ **GET** - Obtém um recurso existente de acordo com os dados da URL.
  - ❑ **POST** - Cria um novo recurso com os dados enviados na requisição.
  - ❑ **PUT** - Atualiza um recurso com base nos dados enviados na requisição.
  - ❑ **DELETE** - Remove um recurso existente.
- ❑ Observe que esses recursos “equivalem” às operações de cadastro, obtenção, atualização e remoção (CRUD) que as aplicações comumente oferecem.
- ❑ Outros verbos menos utilizados **HEAD, TRACE, OPTIONS...**

# HTTP | Status Code

- ❑ Com uma **URL** e um **Verbo**, um cliente pode iniciar uma comunicação com o servidor.
- ❑ Que vai retornar com um **Status Code** e uma resposta.
- ❑ Esse código de status é importante para interpretar a resposta obtida.
  - ❑ Esse status é um código numérico de 3 dígitos.
  - ❑ Entre 200 e 299 é um status de sucesso.
  - ❑ Entre 300 e 399 é um redirecionamento.
  - ❑ Entre 400 e 499 é um erro na mensagem enviada pelo cliente.
  - ❑ Entre 500 e 599 é um erro no servidor.



# HTTP | Status Code | 2xx: Successful

- ❑ Um retorno 2XX indica que a requisição foi bem sucedida.
- ❑ O retorno mais comum é o **200 OK**.
  - ❑ Por exemplo, ao realizar um GET para um arquivo no servidor o retorno é um 200 OK com o conteúdo do arquivo no corpo da resposta.
- ❑ Outros códigos de status comuns:
  - ❑ **201 Created** - O recurso enviado foi criado com sucesso no servidor.
  - ❑ **202 Accepted** - A requisição foi aceita mas o servidor ainda não tem uma resposta. Utilizada em serviços assíncronos.
  - ❑ **204 No Content** - A requisição foi bem sucedida mas não há resposta.

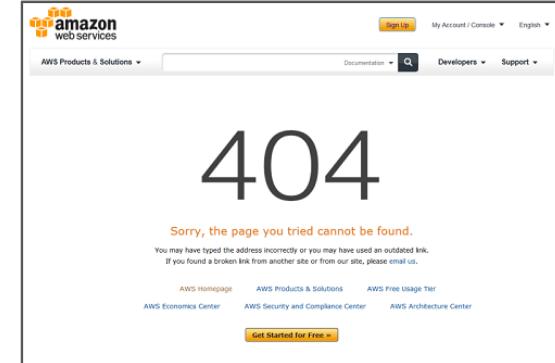


# HTTP | Status Code | 3xx: Redirection

- ❑ Um retorno 3XX indica que uma ação adicional do cliente, para obter o recurso desejado em outro lugar.
- ❑ O retorno mais comum é o **301 Moved Permanently**.
  - ❑ Por exemplo, ao realizar um GET para um arquivo no servidor o retorno é um 300 Moved Permanently e é informada a nova localização do mesmo.
- ❑ Outros códigos de status comuns:
  - ❑ **303 See Other** - O recurso foi temporariamente movido para outra URL.
  - ❑ **304 Not Modified** - O cliente pode enviar dados da versão que possui do recurso desejado, caso ainda seja o mesmo no servidor o retorno é 304.

# HTTP | Status Code | 4xx: Client Error

- ❑ Um retorno 4XX indica que há um problema na requisição que impede que o servidor a processe adequadamente.
- ❑ O retorno mais comum é o **404 Not Found**.
  - ❑ Por exemplo, ao realizar um GET para um recurso não existente no servidor, o retorno é um 404 Not Found.
- ❑ Outros códigos de status comuns:
  - ❑ **400 Bad Request** - A requisição está incompleta ou mal formatada.
  - ❑ **401 Unauthorized** - O cliente não possui credenciais válidas.
  - ❑ **403 Forbidden** - Acesso negado.
  - ❑ **405 Method Not Allowed** - Aquele recurso não suporta o verbo HTTP enviado pelo cliente.



# HTTP | Status Code | 5xx: Server Error

- ❑ Um retorno 5XX indica que o servidor falhou enquanto processava a requisição.
- ❑ O retorno mais comum é o **500 Internal Server Error**.
  - ❑ Por exemplo, ao realizar um GET para um recurso e houve um erro ao acessar o banco de dados, o retorno é um 500 Internal Server Error.
- ❑ Outros códigos de status comuns:
  - ❑ **501 Not Implemented** - A requisição está sendo feita para um recurso que ainda não está pronto para receber requisições.
  - ❑ **503 Service Unavailable** - O servidor está temporariamente fora do ar.



# | HTTP | Comunicação



**Cliente**



**Servidor**

# HTTP | Cabeçalhos

- ❑ Além das URLs, Verbos e Status Code, um importante componente da comunicação HTTP são os cabeçalhos.
- ❑ Eles são enviados junto com a requisição e oferecem informações adicionais:
  - ❑ Sobre o conteúdo da requisição/resposta como tipo de arquivo, tamanho, encoding...
  - ❑ Formatos de arquivo e idioma suportados na requisição/resposta.
  - ❑ Informações de autorização.
  - ❑ Dados do browser e do servidor utilizado...

# HTTP | Exemplo - Requisição

```
GET /doc/test.html HTTP/1.1 ————— Request Line
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35
————— A blank line separates header & body
bookId=12345&author=Tan+Ah+Teck ————— Request Message Body
```

The diagram illustrates the structure of an HTTP request message. It is divided into four main sections:

- Request Line:** The first line of the message, containing the method (GET), the resource path (/doc/test.html), and the protocol version (HTTP/1.1).
- Request Headers:** A group of key-value pairs describing the client's environment and requirements. These include Host, Accept, Accept-Language, Accept-Encoding, User-Agent, and Content-Length.
- Request Message Header:** A bracketed group encompassing the Request Line and the Request Headers.
- A blank line separates header & body:** A horizontal line indicating the boundary between the header and the body of the message.
- Request Message Body:** The final section containing the data being requested, in this case, a query string (bookId=12345&author=Tan+Ah+Teck).

# HTTP | Exemplo - Resposta

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

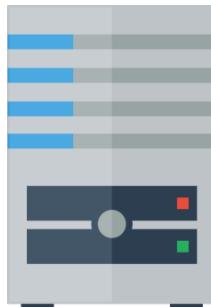
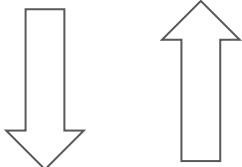
Diagram illustrating the structure of an HTTP response message:

- Status Line:** The first line of the response, containing the protocol version, status code, and reason phrase (e.g., **HTTP/1.1 200 OK**).
- Response Headers:** A group of key-value pairs describing the response, such as Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length, Connection, and Content-Type.
- A blank line separates header & body:** A horizontal line indicating the boundary between the headers and the message body.
- Response Message Body:** The content of the response, which in this example is the HTML document <h1>My Home page</h1>.

# HTTP | Exemplo - Resposta



navegador



Serviço 1

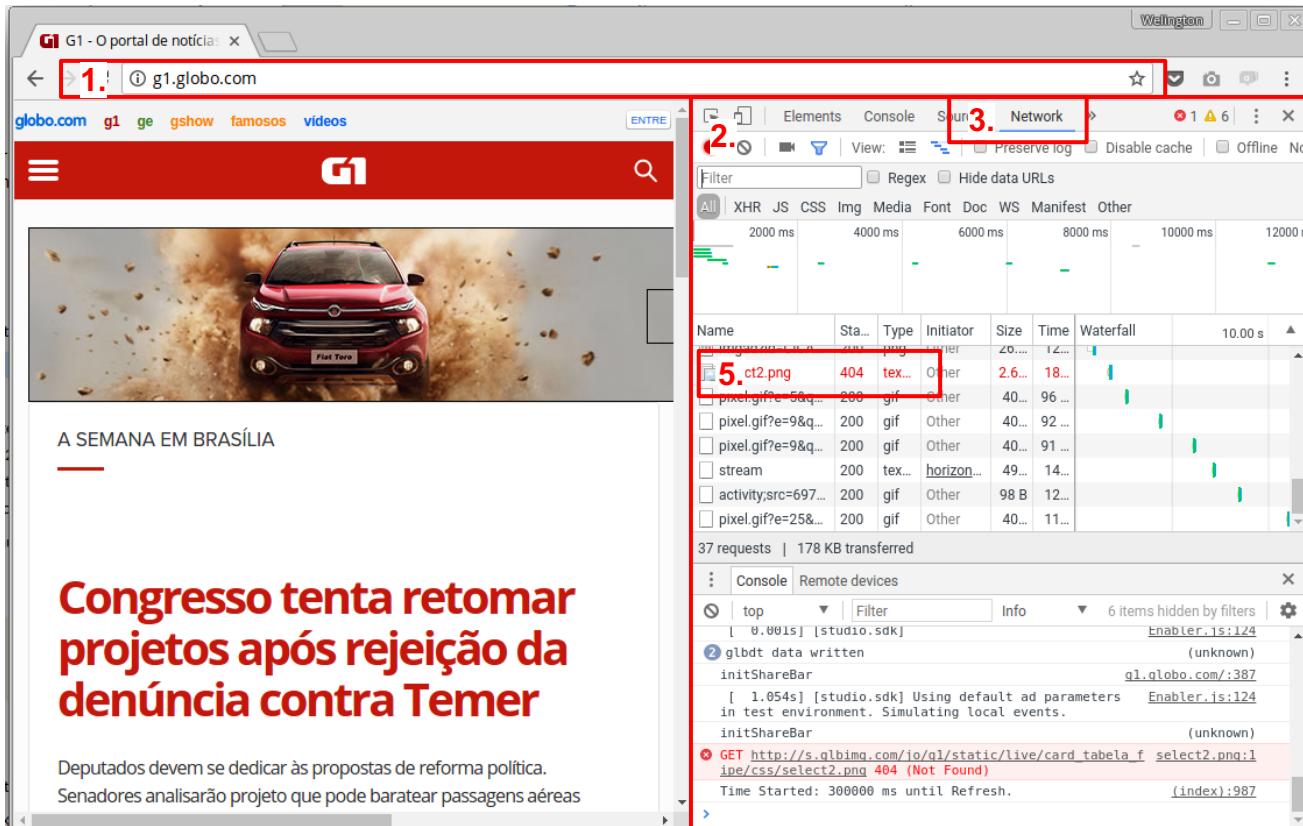
GET /doc/test.html HTTP/1.1  
Host: www.test101.com  
Accept: image/gif, image/jpeg, \*/\*  
Accept-Language: en-us  
Accept-Encoding: gzip, deflate  
User-Agent: Mozilla/4.0  
Content-Length: 35  
  
bookId=12345&author=Tan+Ah+Teck

A blank line separates header & body

HTTP/1.1 200 OK  
Date: Sun, 08 Feb xxxx 01:11:12 GMT  
Server: Apache/1.3.29 (Win32)  
Last-Modified: Sat, 07 Feb xxxx  
ETag: "0-23-4024c3a5"  
Accept-Ranges: bytes  
Content-Length: 35  
Connection: close  
Content-Type: text/html  
  
<h1>My Home page</h1>

# HTTP | Verifique Você Mesmo.

1. Abra o navegador Chrome e navegue até uma página Web.
2. Para abrir as ferramentas de desenvolvedor aperte F12.
3. Selecione a aba Rede (Ou Network).
4. Pressione F5 e veja as requisições HTTP.
5. Selecione uma requisição e observe seus dados e da resposta obtida.



# HTTP | Referências adicionais

1. [https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP\\_Basics.html](https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)
2. <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers>
3. <https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>
4. <https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-2--net-31155>



# Laboratório de Programação de Web Sites Dinâmicos

HTML e páginas dinâmicas

Tassio Sirqueira – 2019/02

# Servlets | Exibindo Informações na Web

- ❑ A única linguagem que os navegadores conseguem interpretar para a exibição de conteúdo é o HTML.
  - ❑ HTML (HyperText Markup Language), que significa Linguagem de Marcação de Hipertexto
- ❑ A partir de declarações HTML podemos definir a estrutura de um documento.
  - ❑ Títulos
  - ❑ Cabeçalho
  - ❑ Barra de navegação
  - ❑ Rodapé
  - ❑ Conteúdo...



# Servlets | Exibindo Informações na Web | Exemplo

## ❑ Exemplo:

- ❑ Suponha que você decide responder com o seguinte texto a solicitação de um navegador.



Mirror Fashion.

Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.

Confira nossas promoções.

Receba informações sobre nossos lançamentos por email.

Navegue por todos nossos produtos em catálogo.

Compre sem sair de casa.

# Servlets | Exibindo Informações na Web | Exemplo

Mirror Fashion.

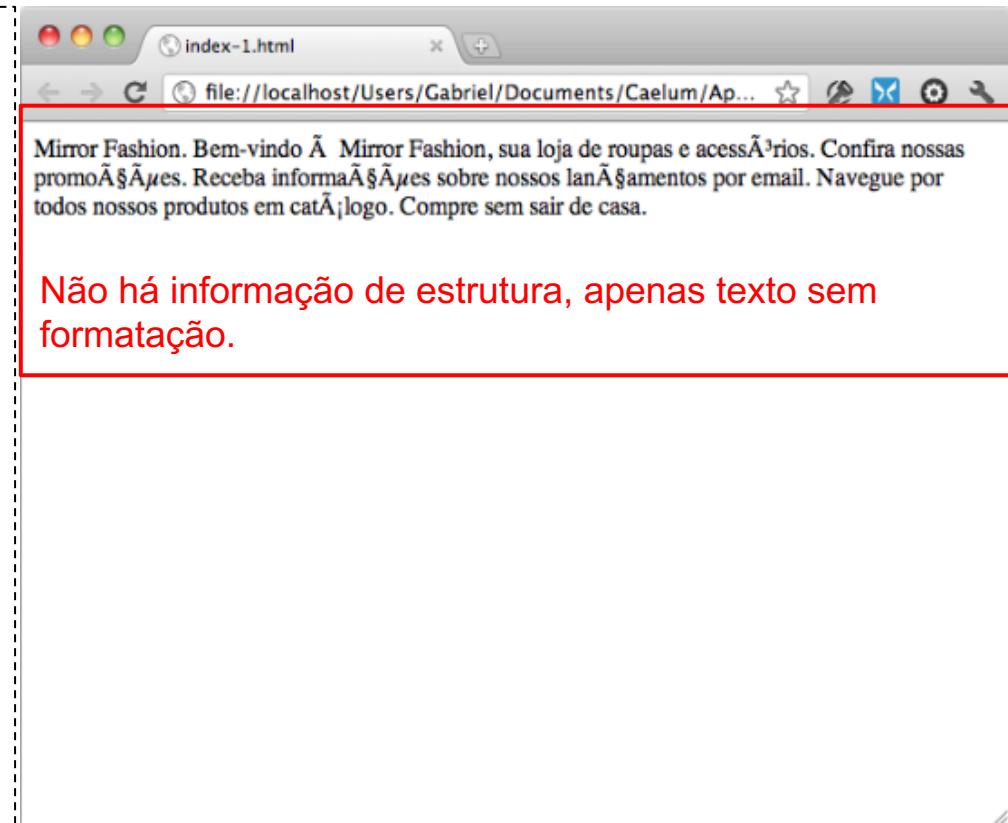
Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.

Confira nossas promoções.

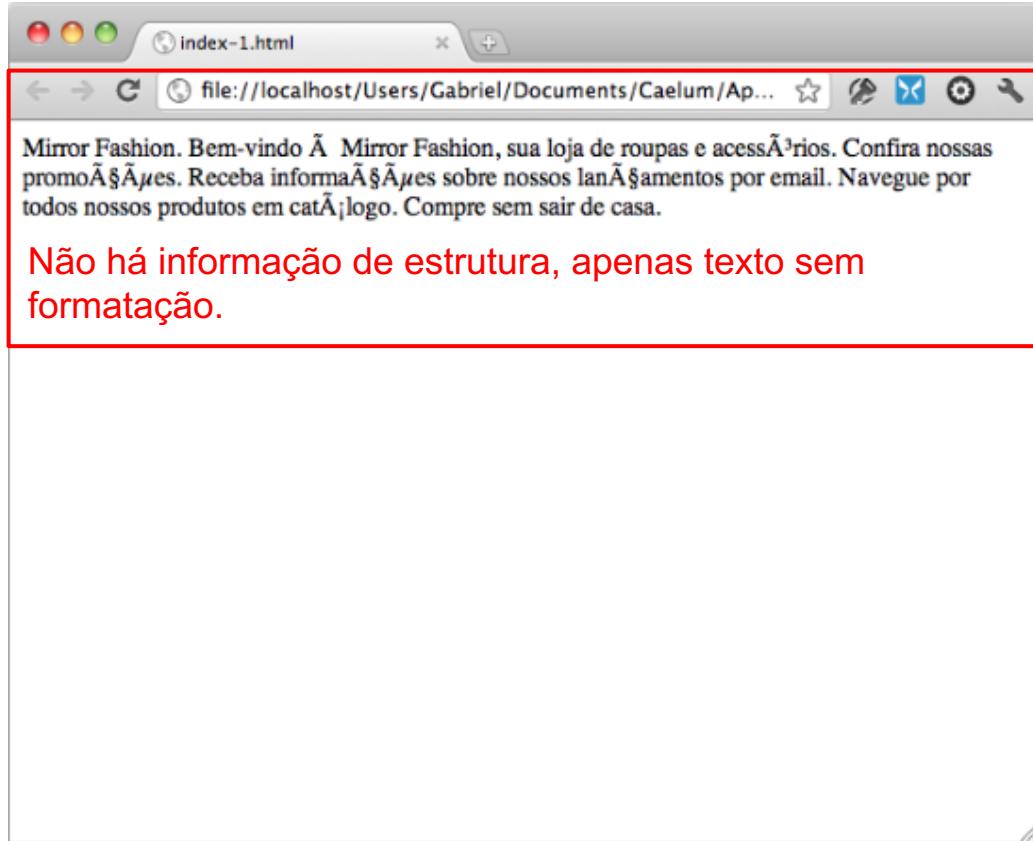
Receba informações sobre nossos lançamentos por email.

Navegue por todos nossos produtos em catálogo.

Compre sem sair de casa.



# Servlets | Exibindo Informações na Web | Exemplo



- Os acentos não são exibidos corretamente.
- Não há quebra de linha.
- Não há marcação de título.
- Não há estrutura para lista.

O HTML permite que a estrutura do documento seja definida!



# Servlets | Exibindo Informações na Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas
e acessórios.</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos
por email.</li>
      <li>Navegue por todos nossos produtos em
catálogo.</li>
      <li>Compre sem sair de casa.</li>
    </ul>
  </body>
</html>
```

Temos uma diretiva especial para indicar ao navegador a versão do HTML que deve ser utilizada.

A diretiva **<!DOCTYPE html>** indica que deve ser utilizada a versão corrente, nesse caso a versão HTML5.



# Servlets | Exibindo Informações na Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas
e acessórios.</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos
por email.</li>
      <li>Navegue por todos nossos produtos em
catálogo.</li>
      <li>Compre sem sair de casa.</li>
    </ul>
  </body>
</html>
```

Todo o documento é definido por marcações, chamadas de **tags**.

Toda tag tem o formato é iniciada por **<tag>** e encerrada por **</tag>**.

As páginas HTML possuem uma tag “pai” que deve estar presente em todo o documento, que é a tag **<html>**.

# Servlets | Exibindo Informações na Web



```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios!</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos por email.</li>
      <li>Navegue por todos nossos produtos em catálogo.</li>
      <li>Compre sem sair de casa.</li>
    </ul>
  </body>
</html>
```

A estrutura de uma página HTML é dividida em **cabeçalho (head) e corpo (body)**.

O cabeçalho de uma página é definido pela tag **<head>** e encerrada por **</head>**. Dentro dessa tag temos apenas metadados e declarações sobre o documento não visíveis na página.

O corpo de uma página é definido pela tag **<body>** e encerrada por **</body>**. Aqui estão todos os elementos exibidos na página.

# Servlets | Exibindo Informações na Web



```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos por email.</li>
      <li>Navegue por todos nossos produtos em catálogo.</li>
      <li>Compre sem sair de casa.</li>
    </ul>
  </body>
</html>
```

Dentro do cabeçalho, temos a tag **<title>** que define o título da página, exibido na barra de títulos ou na aba em que o documento está aberto.

# Servlets | Exibindo Informações na Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h1>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupa
      <ul>
        <li>Confira nossas promoções.</li>
        <li>Receba informações sobre nossos lançamentos
        <li>Navegue por todos nossos produtos em catálo
        <li>Compre sem sair de casa.</li>
      </ul>
    </body>
  </html>
```

No cabeçalho está definida a tag **<meta>** é utilizada para adicionar informações importantes que não são utilizadas na página.

Exemplos são: autor, data de modificação, descrição, palavras chave...

Para adicionar informações em algumas tags, utilizamos o conceito de **atributos**. Nesse caso declaramos o atributo **charset** com valor UTF-8.

Essa declaração define para o navegador o conjunto de caracteres utilizado na página, e corrige os acentos.



# Servlets | Exibindo Informações na Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion.</h2>
    <h2>Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.</h2>
    <ul>
      <li>Confira nossas
      <li>Receba informações
      <li>Navegue por todos
      <li>Compre sem sair de
    </ul>
  </body>
</html>
```

No corpo da página temos várias tags para marcar estrutura.

Entre elas temos os títulos, sub-títulos, títulos de seção...

O HTML possui 5 tags para indicar títulos e subtítulos em diversos níveis:  
**<h1>, <h2>, <h3>, <h4> e <h5>.**

O h1 é o título principal e o h5 o menor nível de subtítulo.



# Servlets | Exibindo Informações na Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mirror Fashion</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Mirror Fashion</h1>
    <h2>Bem-vindo ao Mirror</h2>
    <ul>
      <li>Confira nossas promoções.</li>
      <li>Receba informações sobre nossos lançamentos por email.</li>
      <li>Navegue por todos nossos produtos em catálogo.</li>
      <li>Compre sem sair de casa.</li>
    </ul>
  </body>
</html>
```

Para exibir listas estruturadas temos tags particularmente úteis no HTML.

O **<ul>** permite a declaração de listas não ordenadas.

Cada item da lista é definido pela tag **<li>**.

# Servlets | Exibindo Informações na Web | Exemplo

<title> </title>

Observe que os acentos  
estão preservados  
devido ao  
<meta charset="utf-8">

The screenshot shows a web browser window titled "Mirror Fashion". The URL in the address bar is "file:///localhost/Users/Gabriel/Documents/Caelum/Ap...". The page content is as follows:

```
<title> </title>


# Mirror Fashion.



# Bem-vindo à Mirror Fashion, sua loja de roupas e acessórios.



- Confira nossas promoções. <|i> </i>
- Receba informações sobre nossos lançamentos por email. <|i> </i>
- Navegue por todos nossos produtos em catálogo. <|i> </i>
- Compre sem sair de casa. <|i> </i>


<body> </body>
```

The code segments are highlighted with colored boxes: the title tag is red, the first h1 tag is green, the welcome message is pink, the list items are purple, and the entire body tag is blue.

# Servlets | Outras tags importantes!

## □ Parágrafos

```
<p>Um parágrafo de texto.</p>  
<p>Outro parágrafo de texto.</p>
```

## □ Ênfase

```
<p>Compre suas roupas e acessórios na <strong>Mirror Fashion</strong>.
```

## □ Links

```
<p> Visite<a href="http://www.site.com.br">o site</a>. </p>
```

## □ Imagens

```

```

## □ Alguma área na estrutura da página

```
<div> ... </div>
```

# Servlets | Outras tags importantes! | Formulários

## □ Caixas de texto

```
Nome: <input type="text" name="fname">  
E-mail: <input type="email" name="email_usuario" />  
Receber newsletter: <input type="checkbox" name="news" />  
E-mail público <input type="checkbox" name="publico" />
```

## □ Botões

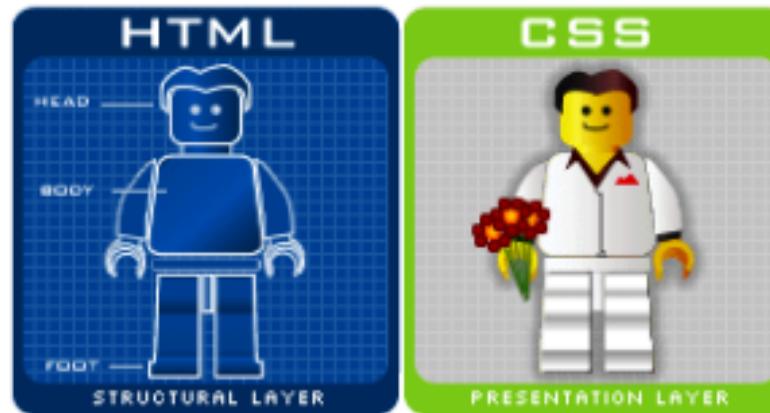
```
<button name="button">Click me</button>
```

## □ Formulários

```
<form action="/nomes" method="post">  
  Nome: <input type="text" name="nome">  
  <input type="submit" value="Submit">  
</form>
```

# Servlets | E como mudar a aparência da minha página?

- ❑ A finalidade da linguagem HTML é definir a estrutura do documento.
- ❑ Para definir a aparência ou o “estilo” de um documento utilizamos o CSS.
  - ❑ Cascading Style Sheets (CSS) - Folhas de Estilo em Cascata.



# Servlets | E como mudar a aparência da minha página?

- Uma outra forma de estilizar um documento, e mais recomendada, é usando um “seletor” para alterar todos os elementos com determinada característica em uma página.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre a Mirror Fashion</title>
    <style>
      p {
        background-color: yellow;
        color: blue;
      }
    </style>
  </head>
  <body>
    <p> O conteúdo desta tag será exibido em azul com fundo amarelo! </p>
  </body>
</html>
```

O seletor **nome-da-tag** aplica as regras de estilo para todos os elementos daquela tag na página.

# Servlets | E como mudar a aparência da minha página?

- Uma outra forma de definir o que deve ser exibido é através de classes.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre a Mirror Fashion</title>
    <style>
      .destaque {
        background-color: yellow;
        color: blue;
      }
    </style>
  </head>
  <body>
    <p class="destaque"> O conteúdo desta tag será exibido em azul com fundo amarelo! </p>
    <p> O conteúdo desta tag será preto. </p>
  </body>
</html>
```

O seletor **.classe** aplica as regras de estilo para todos os elementos com determinada classe na página.

Para definir uma classe em um elemento utilize o atributo **class**.

# Servlets | E como mudar a aparência da minha página?

- Arquivos de estilo podem ainda ser importados de outros arquivos.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Sobre a Mirror Fashion</title>
    <link rel="stylesheet" href="estilos.css">
  </head>
  <body>
    <p class="destaque"> O conteúdo desta tag será exibido
em azul com fundo amarelo! </p>
    <p> O conteúdo desta tag será preto. </p>
  </body>
</html>
```

A tag **link** importa o arquivo cuja URI  
está definida pelo atributo **href**.

```
/* conteúdo do arquivo estilos.css */
.destaque {
  background-color: yellow;
  color: blue;
}
```

# Servlets | Outras propriedades de estilo importantes!

## Dimensões

```
height: 300px;  
width: 300px;
```

## Espaçamento

```
padding: 10px 20px 15px 5px;
```

## Borda

```
border: 1px 2px 1px 2px;
```

## Alinhar à esquerda ou direita

```
float: right;
```

## Centralizar horizontalmente.

```
margin: 0 auto;
```

Cores, fontes, posicionamento, transformações...

# Servlets | E como adicionar comportamentos à minha página?

- ❑ A finalidade da linguagem HTML é definir a estrutura do documento, e do CSS é definir a aparência do mesmo.
- ❑ Para adicionar comportamento à página, podemos utilizar JavaScript.
  - ❑ Apesar do nome, a linguagem JavaScript tem pouco a ver com o Java.
- ❑ Suportada por diferentes navegadores.
- ❑ Inicialmente utilizada apenas para scripts simples.
- ❑ Hoje em dia é uma linguagem poderosa e completa.
  - ❑ Diversas aplicações além da web.

JS



# Servlets | E como adicionar comportamentos à minha página?

- Imperativa e Estruturada
  - if, while, do...while no estilo C.
- Baseada em objetos
  - Objetos JavaScript são arrays associativos.
  - Podem ser extendidos por meio de protótipos.
- Funcional
  - Funções são tipos de “primeira classe”
    - Podem ser passados como argumentos, serem atribuídos a variáveis ou retornados como qualquer outro objeto.



# Servlets | E como adicionar comportamentos à minha página?

```
var cars = [{type:"Volvo", year:2016}, {type:"Saab", year:2001},{type:"BMW", year:2010}]

function displayCars() {
    document.getElementById("demo").innerHTML =
    cars[0].type + " " + cars[0].year + "<br>" +
    cars[1].type + " " + cars[1].year + "<br>" +
    cars[2].type + " " + cars[2].year;
}

function funcao() {
    cars.sort(function(a, b){return a.year - b.year});
    displayCars();
}

funcao();
```

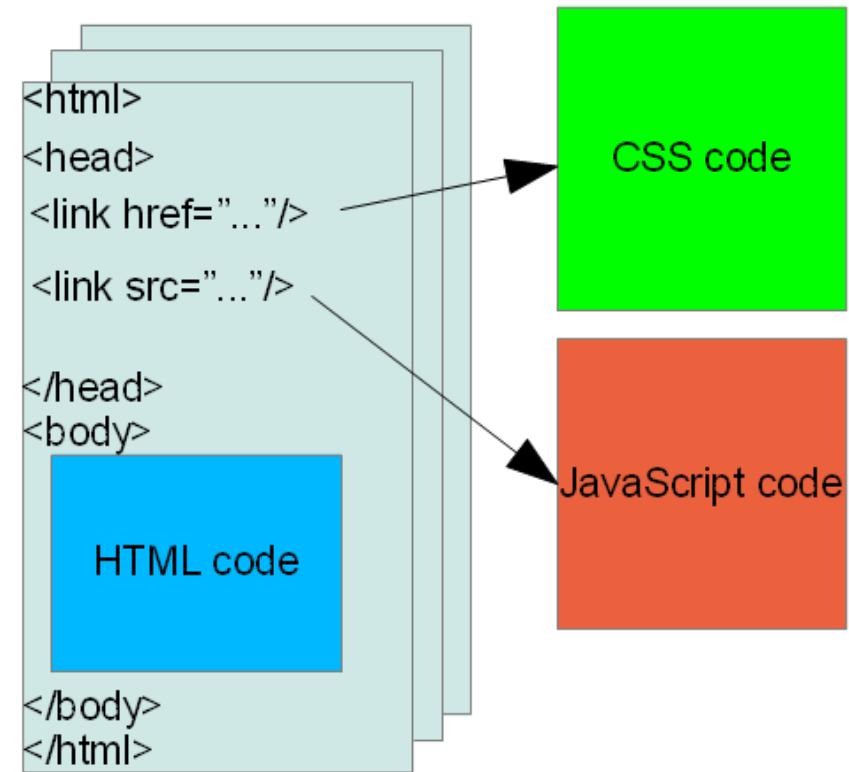
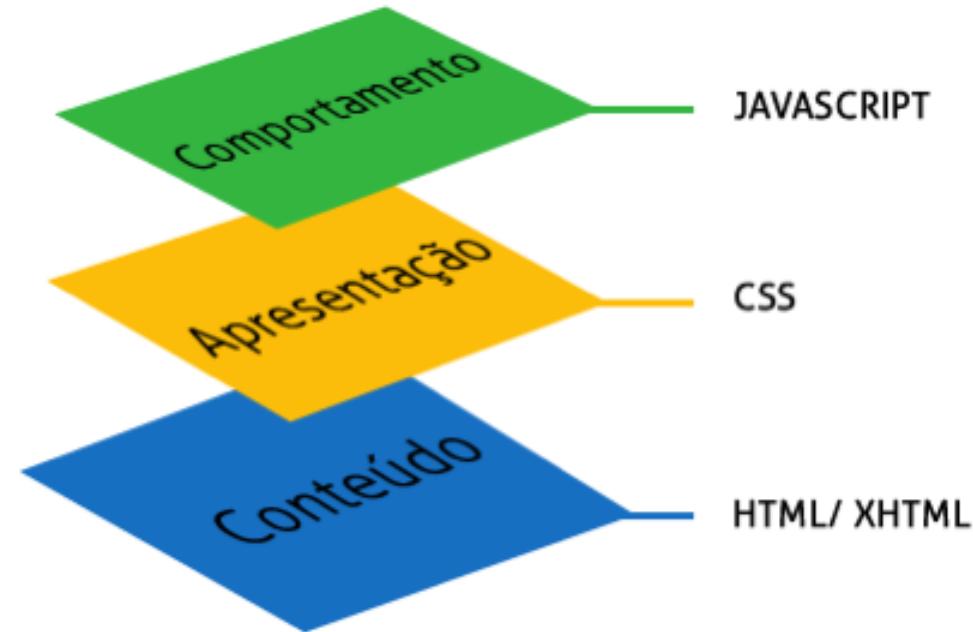


# Servlets | E como adicionar comportamentos à minha página?

- ❑ JavaScript é poderoso porque permite que código seja executado no lado cliente.
  - ❑ Seu código pode rodar dentro do browser, na máquina do cliente.
- ❑ No entanto existem várias necessidades reais de aplicações web que não podem ser resolvidas dessa forma.
  - ❑ Salvamento de dados em um banco de dados.
  - ❑ Regras de negócio e validações.
  - ❑ Autenticação e autorização.
- ❑ O código JavaScript deve
  - ❑ Oferecer uma página mais dinâmica e agradável para navegação.
  - ❑ Melhorar a experiência do usuário por meio de interfaces e dinâmicas validações.



# Servlets | E como adicionar comportamentos à minha página?



# Prática | Disponibilizando uma página Web Estática no Glassfish

- ❑ Do que eu preciso para começar?
- ❑ **JDK** = Java Development Kit
  - ❑ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Sugiro fazer download do netbeans separado.

Java SE Downloads

Java Platform (JDK) 8u144

Java Platform, Standard Edition

Java SE 8u144

Java EE

Java ME

Java ME Support

Java SE Advanced & Suite

Java Embedded

Java DB

Web Tier

Java Card

Java TV

New to Java

Community

Java Magazine

Overview Downloads Documentation

DOWNLOAD +

NetBeans

DOWNLOAD +

Java Platform (JDK) 8u144

Java APIs

Technical Articles

Demos and Videos

Java Resources

Java SE

Java EE

Java ME

Java Support

Java SE Advanced & Suite

Java Embedded

Java DB

Web Tier

Java Card

Java TV

New to Java

Community

Java Magazine

Technology

Documentation

Community

Technologies

Training

Oracle Technology Network > Java > Java SE > Downloads

Menu

Sign In

Country

Call

Java EE and Glassfish

Java ME

Java Card

NetBeans IDE

Java Mission Control

Java APIs

Technical Articles

Demos and Videos

Forums

Java Magazine

Java.net

Developer Training

Tutorials

Java.com

Para usuários de Debian / Ubuntu / etc.. sugiro a instalação via PPA.

Java SE Development Kit 8 Downloads

Java SE Development Kit 8u144

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement  Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	<a href="#">jdk-8u144-linux-arm32-vfp-hf.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.83 MB	<a href="#">jdk-8u144-linux-arm64-vfp-hf.tar.gz</a>
Linux x86	164.65 MB	<a href="#">jdk-8u144-linux-i586.rpm</a>
Linux x86	179.44 MB	<a href="#">jdk-8u144-linux-i586.tar.gz</a>
Linux x64	162.1 MB	<a href="#">jdk-8u144-linux-x64.rpm</a>
Linux x64	176.92 MB	<a href="#">jdk-8u144-linux-x64.tar.gz</a>
Mac OS X	226.6 MB	<a href="#">jdk-8u144-macosx-x64.dmg</a>
Solaris SPARC 64-bit	139.87 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	99.18 MB	<a href="#">jdk-8u144-solaris-sparcv9.tar.gz</a>
Solaris x64	140.51 MB	<a href="#">jdk-8u144-solaris-x64.tar.Z</a>
Solaris x64	96.99 MB	<a href="#">jdk-8u144-solaris-x64.tar.gz</a>
Windows x86	190.94 MB	<a href="#">jdk-8u144-windows-i586.exe</a>

Java APIs

Technical Articles

Demos and Videos

Forums

Java Magazine

Java.net

Developer Training

Tutorials

Java.com

# Prática | Disponibilizando uma página Web Estática no Glassfish

- ❑ Do que eu preciso para começar?
- ❑ IDE Netbeans

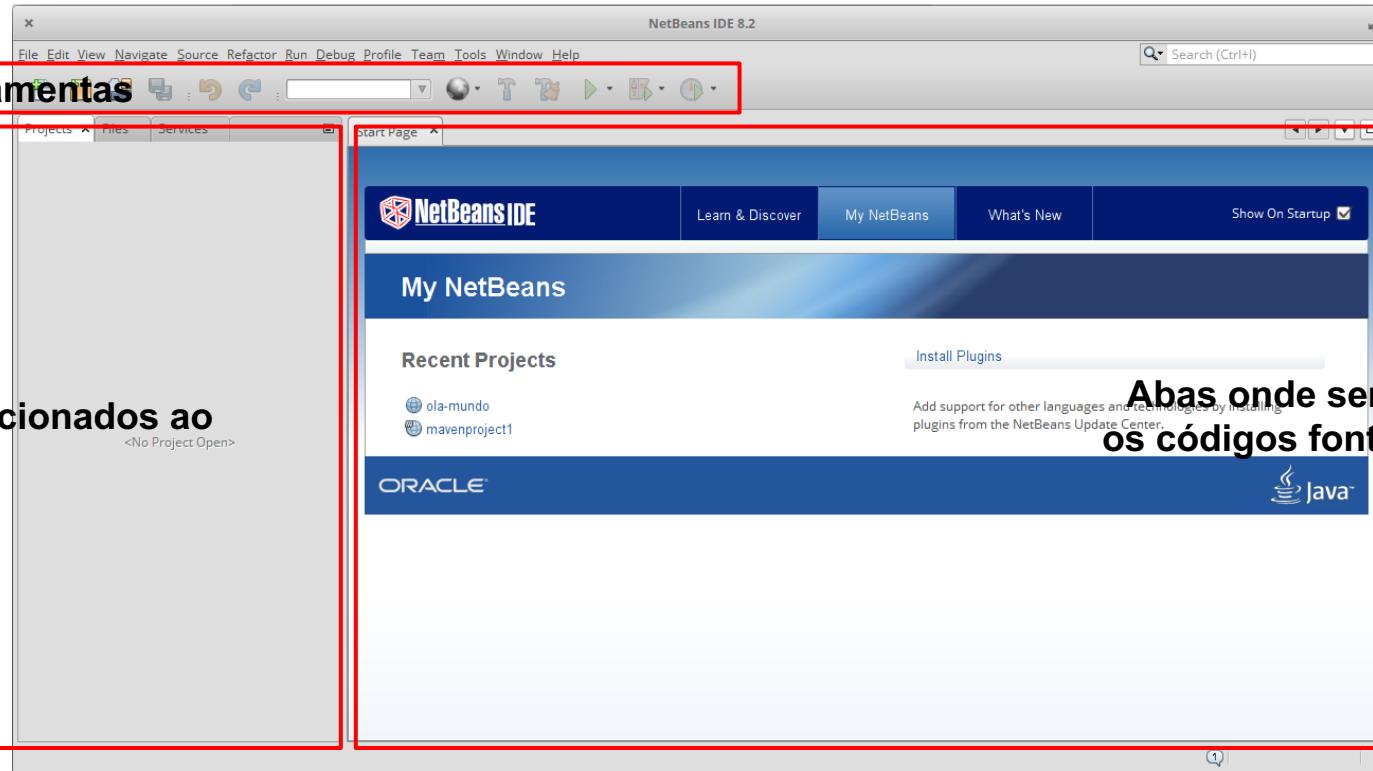
The screenshot shows the NetBeans IDE 8.2 Download page. At the top, there are navigation links for NetBeans IDE, NetBeans Platform, Plugins, Docs & Support, Community, and Partners, along with a search bar. Below the header, it says "HOME / Download". The main title is "NetBeans IDE 8.2 Download" with version options 8.1, 8.2, Development, and Archive. There are fields for Email address (optional), IDE Language (set to English), and Platform (set to Linux (x86/x64)). A note states: "Note: Greyed out technologies are not supported for this platform." On the left, a sidebar lists "Supported technologies" including Java SE, Java EE, HTML5/JavaScript, PHP, C/C++, and others. The "Java EE" row is highlighted with a red box. To the right, a grid titled "NetBeans IDE Download Bundles" shows the availability of different technologies across various platforms. The Java EE column is highlighted with a red box. At the bottom, there are download links for Java SE, Java EE, HTML5/JavaScript, PHP, C/C++, and All, with specific links for x86, x64, and Linux platforms.

Supported technologies *	Java SE	Java EE	HTML5/JavaScript	PHP	C/C++	All
NetBeans Platform SDK	•	•				•
Java SE	•	•				•
Java FX	•	•				•
Java EE		•				•
Java ME						•
HTML5/JavaScript		•	•	•		•
PHP			•	•		•
C/C++					•	•
Groovy						•
Java Card™ 3 Connected						—
Bundled servers						
GlassFish Server Open Source Edition 4.1.1		•				•
Apache Tomcat 8.0.27		•				•

Free, 94 MB      Free, 196 MB      Free, 116 - 119 MB      Free, 116 - 119 MB      Free, 115 - 117 MB      Free, 214 MB

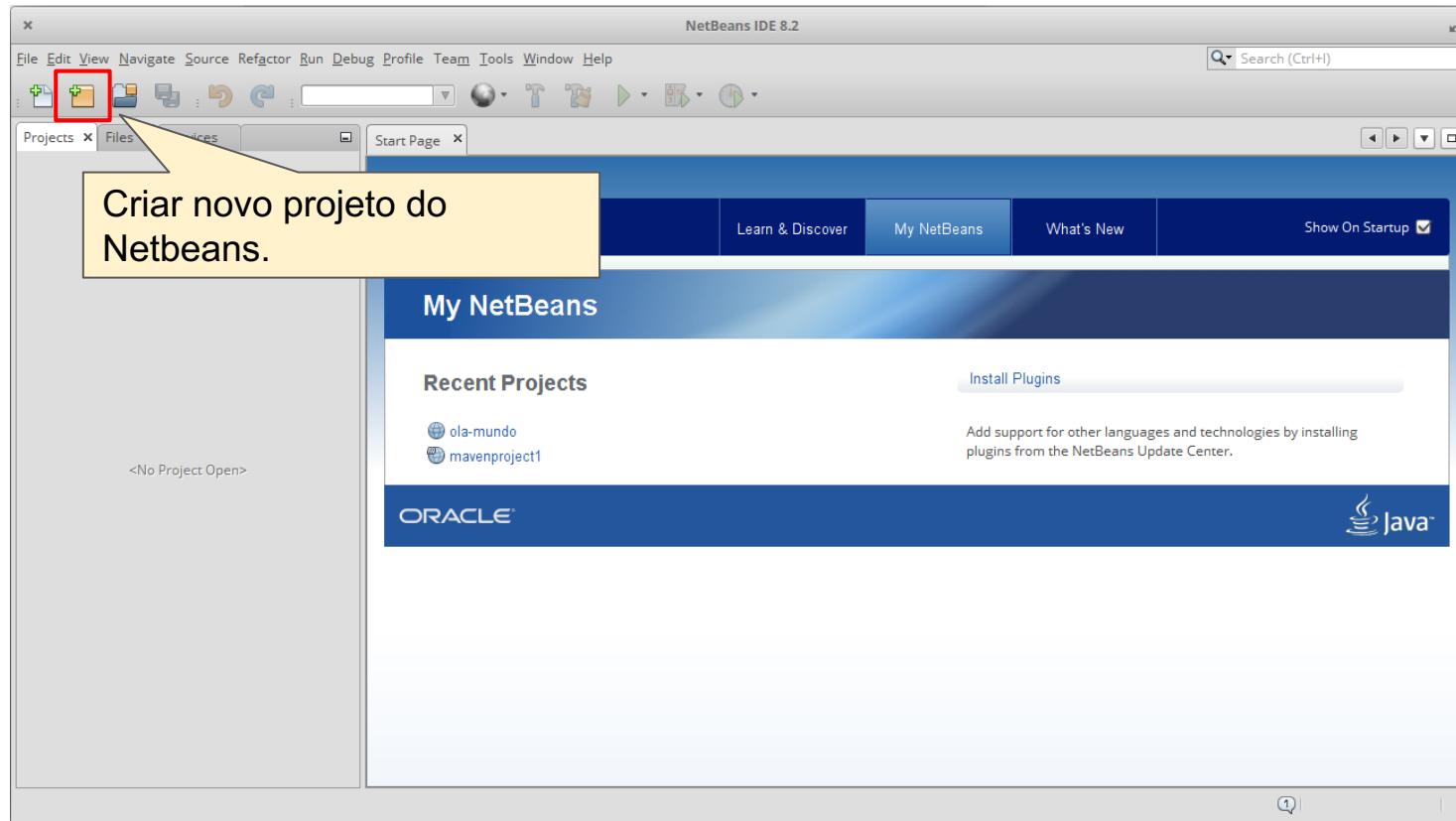
# Prática | Disponibilizando uma página Web Estática no Glassfish

- Instale o JDK e o NetBeans, durante a instalação não desmarque a instalação conjunta do Glassfish.



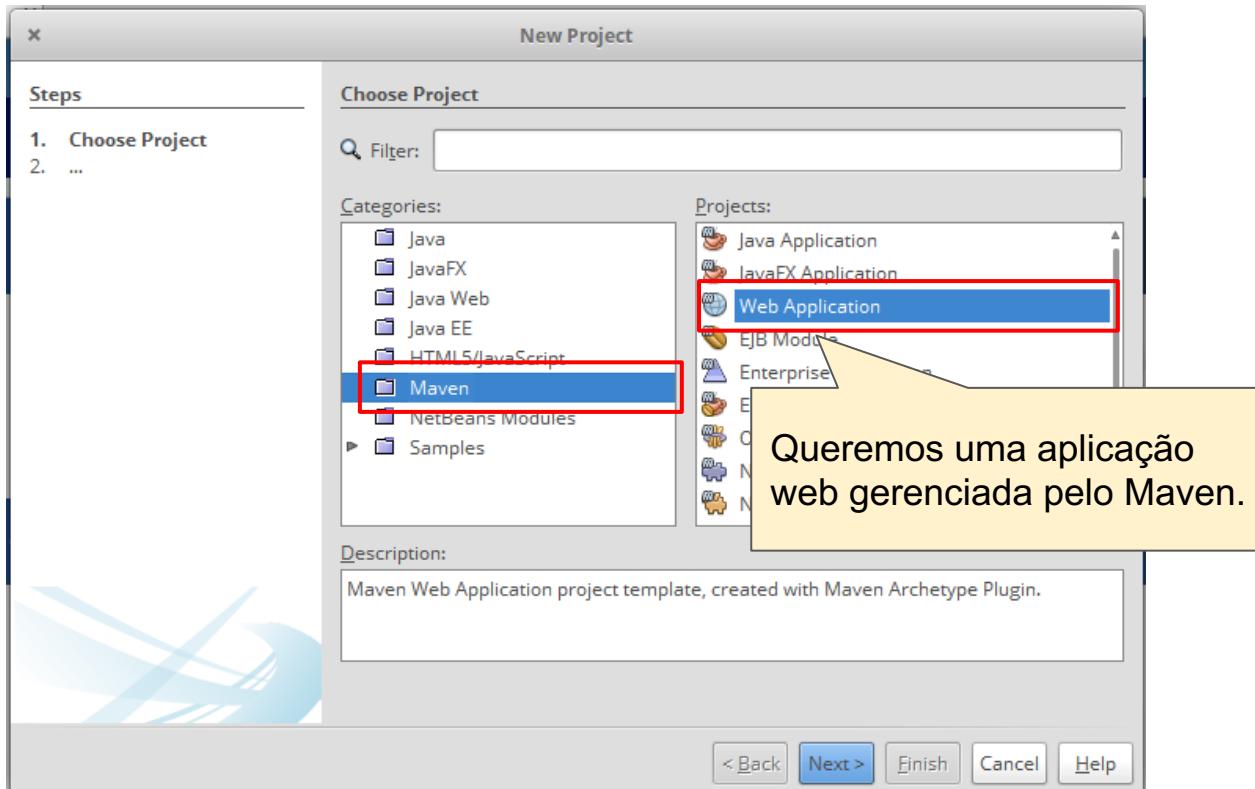
# Prática | Disponibilizando uma página Web Estática no Glassfish

- Crie novo projeto a partir do ícone indicado.



# Prática | Disponibilizando uma página Web Estática no Glassfish

- Escolha o tipo de projeto.



# Prática | Disponibilizando uma página Web Estática no Glassfish

- ❑ Apache Maven ( Maven) é uma ferramenta de automação de compilação.
  - a. Permite o gerenciamento de dependências facilitado, sem necessidade de baixar bibliotecas e suas dependências diretamente.
  - b. Permite a configuração da compilação.
  - c. Facilita o gerenciamento de configuração do projeto.
  - d. Extensível por meio de uma variedade de plugins.
  - e. Deixa o projeto independente da IDE.
- ❑ Se você criar um projeto sem Maven todo o código funcionará normalmente.
  - a. Porém gerenciado pelo Netbeans.



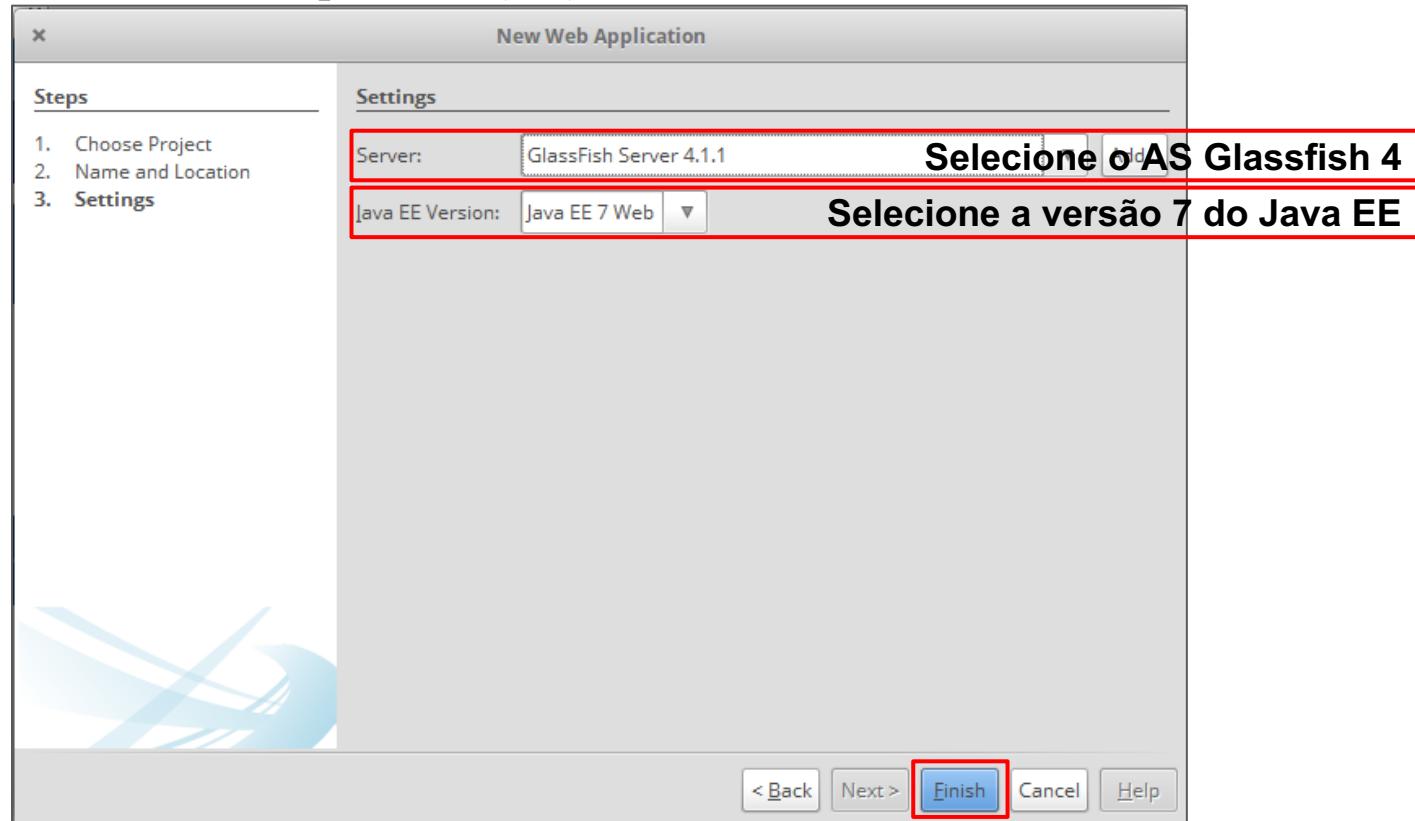
# Prática | Disponibilizando uma página Web Estática no Glassfish

## □ Parâmetros do projeto

New Web Application	
<b>Steps</b>	<b>Name and Location</b>
1. Choose Project 2. Name and Location 3. Settings	<b>Project Name:</b> exemplo <b>Nome do projeto</b> <b>Project Location:</b> /home/welington/Workspace/professor-netbeans <b>Pasta onde será criado</b> <b>Project Folder:</b> /home/welington/Workspace/professor-netbeans/exemplo  <b>Artifact Id:</b> exemplo  <b>Group Id:</b> com.mycompany <b>Nome da organização</b> <b>Version:</b> 1.0-SNAPSHOT <b>Versão</b> <b>Base Package:</b> com.mycompany.exemplo (Optional) <b>Pacote base</b>
Em Java, pacotes e group ids são por padrão o DNS reverso da organização. Ex: <ul style="list-style-type: none"><li>□ google.com<ul style="list-style-type: none"><li>□ Com.google</li></ul></li><li>□ globoesporte.g1.com.br<ul style="list-style-type: none"><li>□ Br.com.g1.globoesporte</li></ul></li><li>□ www.cesjf.br<ul style="list-style-type: none"><li>□ br.cesjf</li></ul></li></ul>	<b>&lt; Back</b> <b>Next &gt;</b> <b>Finish</b> <b>Cancel</b> <b>Help</b>

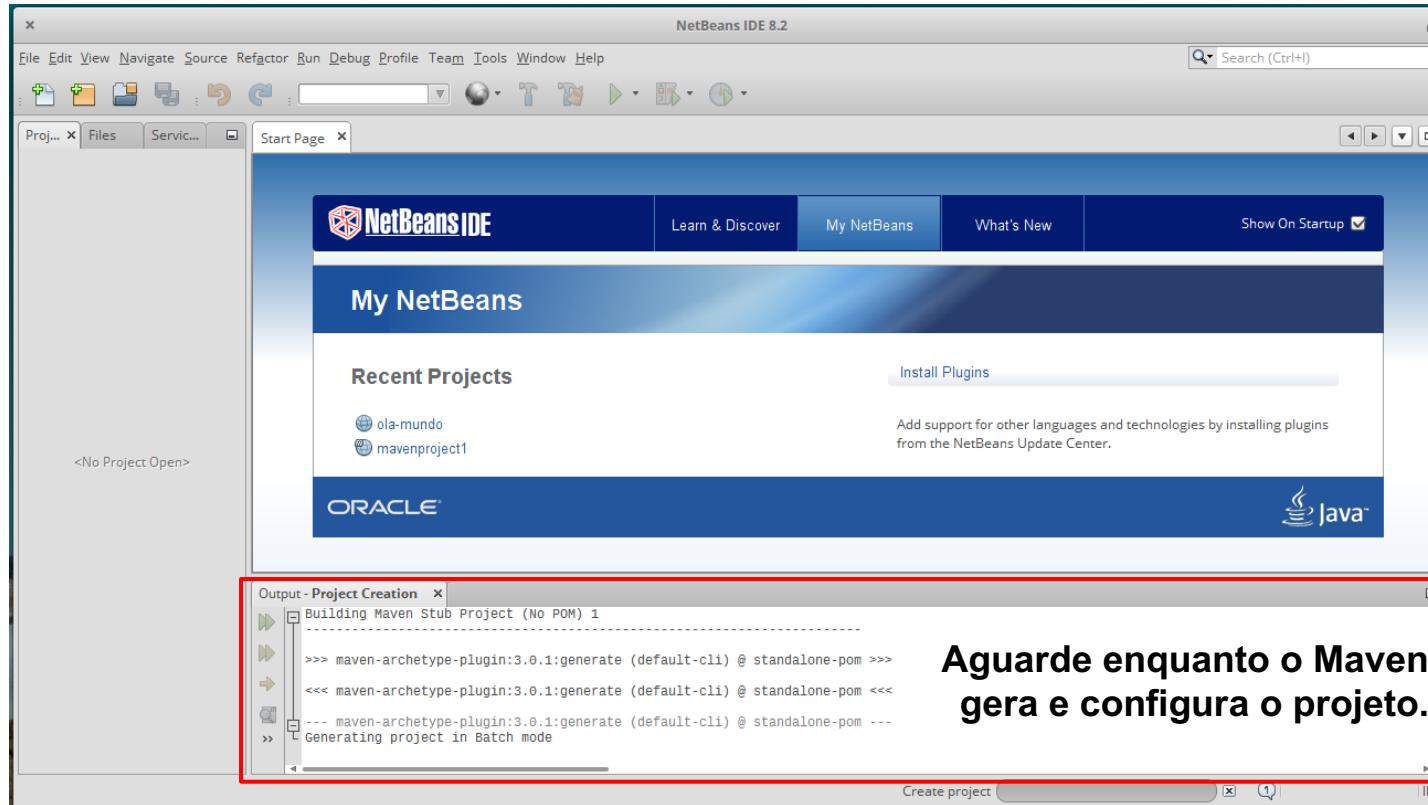
# Prática | Disponibilizando uma página Web Estática no Glassfish

## ☐ Configuração do Servidor de Aplicação (AS)



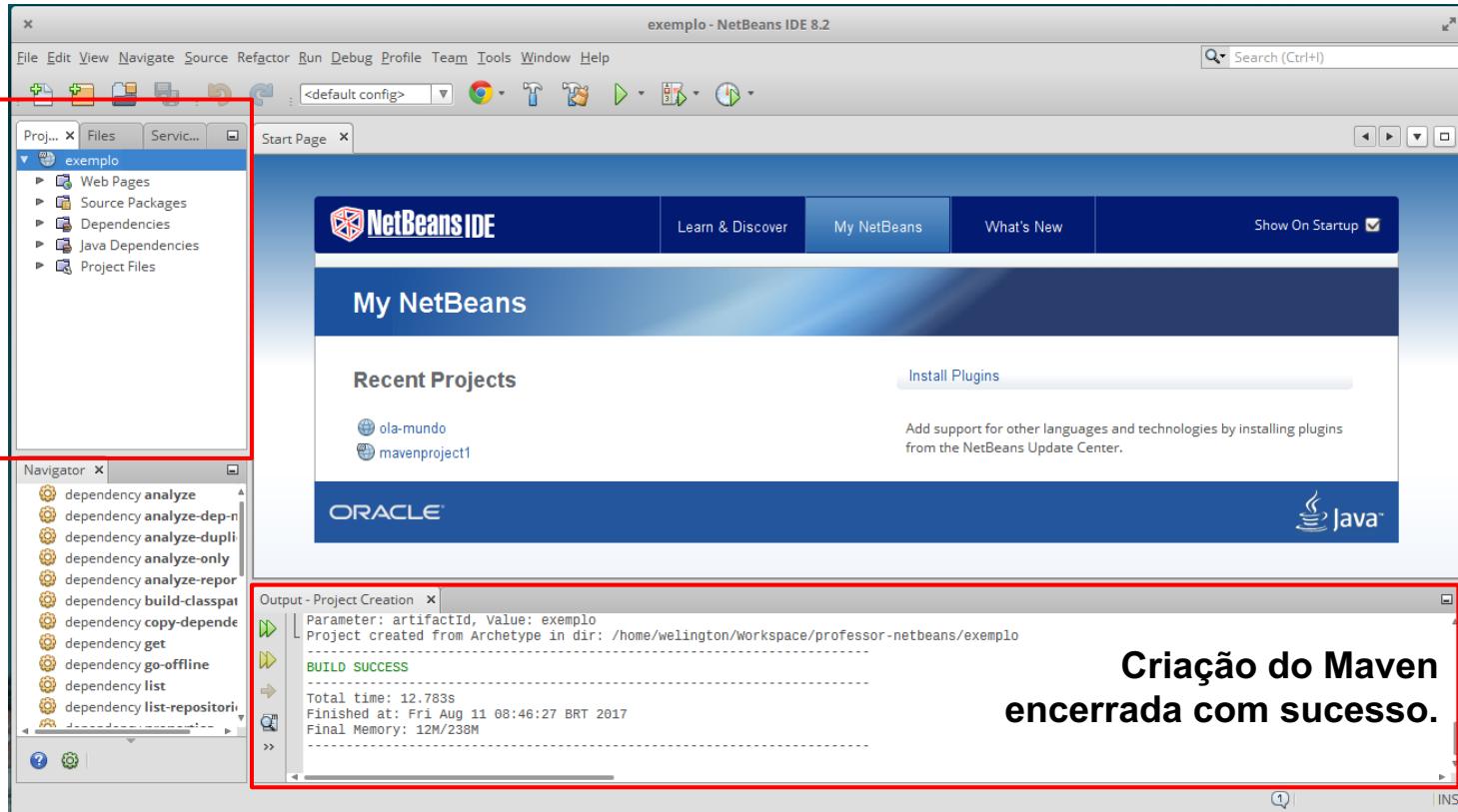
# Prática | Disponibilizando uma página Web Estática no Glassfish

## □ Parâmetros do projeto



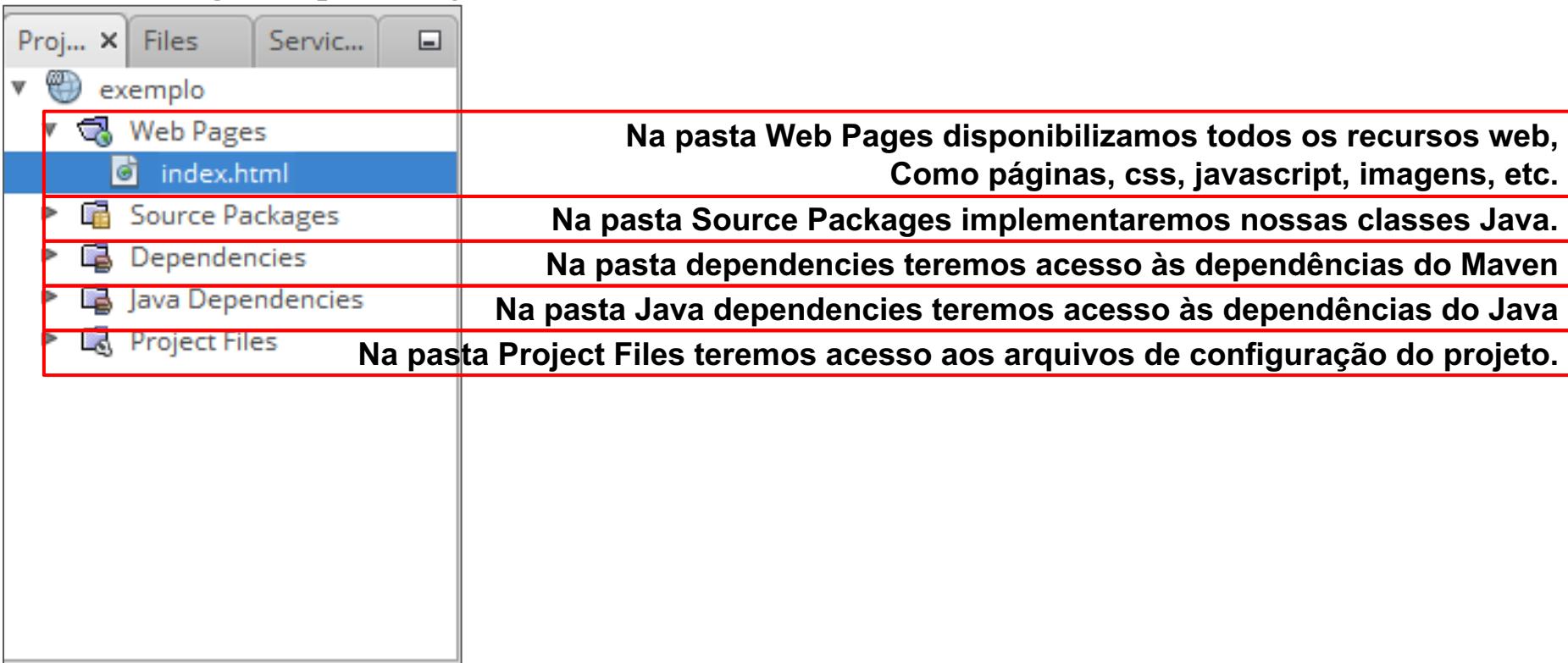
# Prática | Disponibilizando uma página Web Estática no Glassfish

## □ Parâmetros do projeto



# Prática | Disponibilizando uma página Web Estática no Glassfish

## □ Navegando pelo Projeto



# Prática | Disponibilizando uma página Web Estática no Glassfish

- ❑ Rodando o projeto.

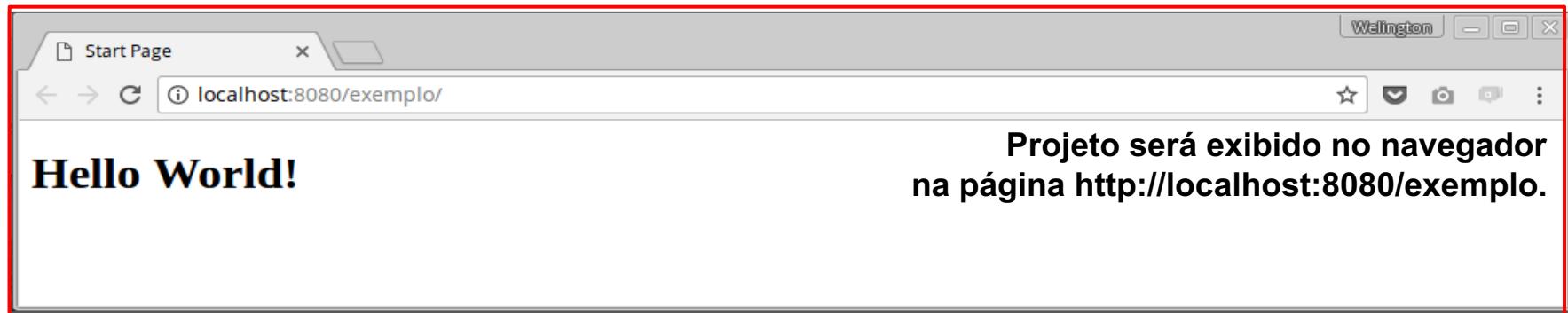


- ❑ Verificando a saída do glassfish e da execução do projeto pelo maven.

```
Output x
Run (exemplo) x Java DB Database Process x GlassFish Server 4.1.1 x
Info: visiting unvisited references
Info: visiting unvisited references
Info: Loading application [exemplo] at [/exemplo]
Info: exemplo was successfully deployed in 1,272 milliseconds.
Warning: Instance could not be initialized. Class=interface org.glassfish.grizzly.http.server.AddOn, name=http-listener-1, realClassName=org.gla
Info: Created HTTP listener http-listener-1 on host/port 0.0.0.0:8080
Info: Grizzly Framework 2.3.23 started in: 1ms - bound to [/0.0.0.0:8080]
```

Endereço em que a aplicação foi disponibilizada. (/exemplo)

Porta em que a aplicação foi disponibilizada. (8080)

The screenshot shows the NetBeans Output window with several tabs: Run (exemplo), Java DB Database Process, and GlassFish Server 4.1.1. The GlassFish tab displays deployment logs. Two specific lines of text are highlighted with red boxes and annotated with text: "Endereço em que a aplicação foi disponibilizada. (/exemplo)" above the line "Info: Loading application [exemplo] at [/exemplo]", and "Porta em que a aplicação foi disponibilizada. (8080)" above the line "Info: Created HTTP listener http-listener-1 on host/port 0.0.0.0:8080".

# HTTP | Referências adicionais

1. <https://www.caelum.com.br/apostila-html-css-javascript/>
2. <https://developer.mozilla.org/pt-BR/docs/Web/HTML>
3. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>
4. <https://developer.mozilla.org/pt-BR/docs/Web/CSS>

# Laboratório de Programação de Web Sites Dinâmicos

Servlets

Tassio Sirqueira – 2019/02

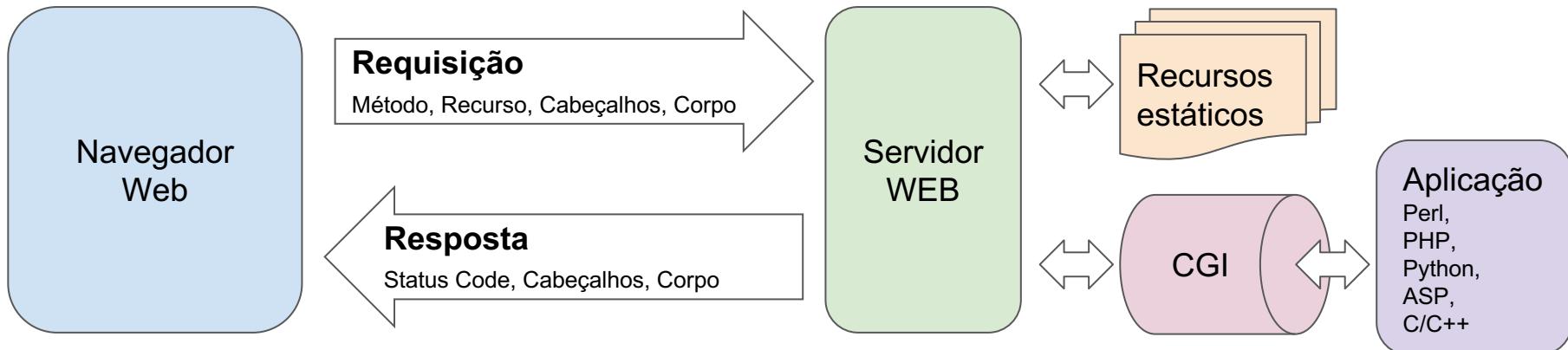
# Servlets | Introdução

- ❑ Quando a web surgiu, o objetivo era realizar a troca de documentos estáticos por meio de HTTP.
  - ❑ Páginas HTML
  - ❑ Imagens
  - ❑ Animações...
- ❑ Logo observou-se o potencial de comunicação da Web.
  - ❑ Páginas estáticas não seriam o suficiente.
- ❑ Era necessário responder de forma dinâmica às requisições dos usuários.
- ❑ Hoje boa parte do que acessamos na Web é baseado em conteúdos dinâmicos.
  - ❑ Portais de notícias
  - ❑ Blogs
  - ❑ Bancos...



# Servlets | Introdução

- ❑ A primeira forma de se gerar páginas dinâmicas foi por meio do CGI.
  - ❑ CGI - Common Gateway Interface
- ❑ Interface padrão a partir da qual o servidor Web executaria um programa externo.
  - ❑ Tipicamente PERL
  - ❑ Outras linguagens começaram a surgir/suportar o padrão, como PHP, ASP, C/C++...



# Servlets | Introdução

- ❑ Na plataforma Java, a primeira e principal tecnologia capaz de gerar páginas dinâmicas são as **Servlets**.
  - ❑ Surgiram no ano de 1997
  - ❑ As versões mais encontradas no mercado são a 2.4 (J2EE 1.4) e a 2.5 (Java EE 5).
  - ❑ A versão atual é a 3.1 ( Java EE 7).
- ❑ Usaremos a própria linguagem Java para isso, criando uma classe que terá capacidade de gerar conteúdo HTML.
- ❑ O nome "servlet" vem da ideia de um pequeno servidor (servidorzinho, em inglês) .

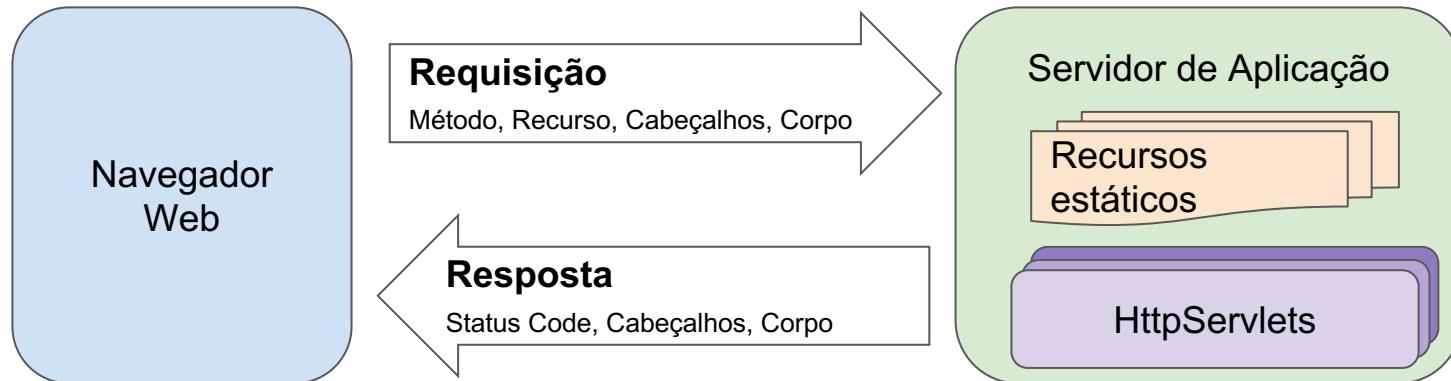
# Servlets | Introdução

## ❑ Servlets x CGI

- ❑ Diversas requisições podem ser feitas à mesma servlet ao mesmo tempo em um único servidor.
- ❑ Fica na memória entre requisições, não precisa ser re-instanciada.
- ❑ O nível de segurança e permissão de acesso pode ser controlado em Java.
- ❑ Em CGI, cada cliente é representado por um processo, enquanto que com Servlets, cada cliente é representado por uma linha de execução (*Thread*).

# Servlets | Introdução

- ❑ Servlets são classes Java que implementam uma interface específica para tratar requisições e respostas.
  - ❑ HttpServlet



# Servlets | Deployment Descriptor

- ❑ Para que possa tratar as requisições HTTP precisamos “dizer” ao servidor web quais são os Servlets do nosso projeto.
- ❑ Para isso utilizamos um arquivo específico o **web.xml**
  - ❑ web.xml é o **Deployment Descriptor** (Descriptor de implantação).
- ❑ Esse arquivo deve ficar em uma pasta especial chamada **WEB-INF** dentro da pasta de recursos do projeto.
  - ❑ Nada dentro de META-INF será acessível via HTTP.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>olamundo</servlet-name>
        <servlet-
class>com.exemplo.controller.OlaMundo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>olamundo</servlet-name>
        <url-pattern>/ola-mundo</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

# Servlets | Introdução

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>olamundo</servlet-name>
        <servlet-
class>com.exemplo.controller.OlaMundo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>olamundo</servlet-name>
        <url-pattern>/ola-mundo</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Qual o nome do Servlet

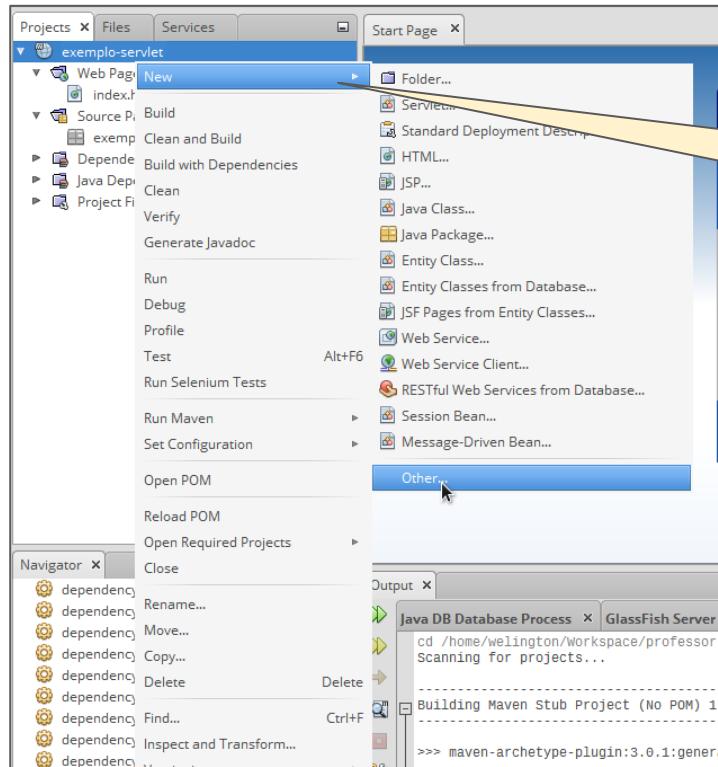
Qual classe implementa o Servlet indicado.

Mapeamos um recurso (URI) a  
Um servlet declarado nesse arquivo.

Existem uma série de configurações possíveis nesse arquivo.  
Entre elas tempo de sessão, bibliotecas de tags customizadas,  
Páginas de erro personalizadas, segurança, etc...

# Servlets | Deployment Descriptor | Netbeans

## □ Criando um web.xml no Netbeans.

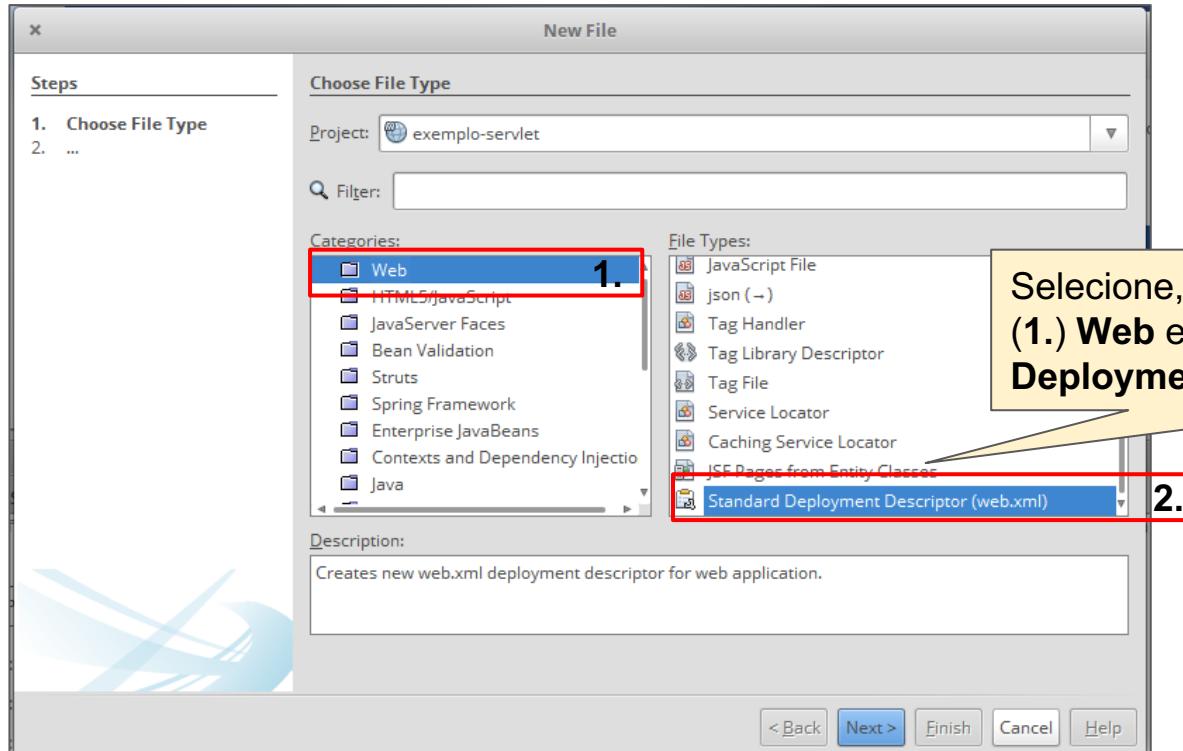


Clique com o botão direito no nome do projeto.

Selecione **New > Other...**

# Servlets | Deployment Descriptor | Netbeans

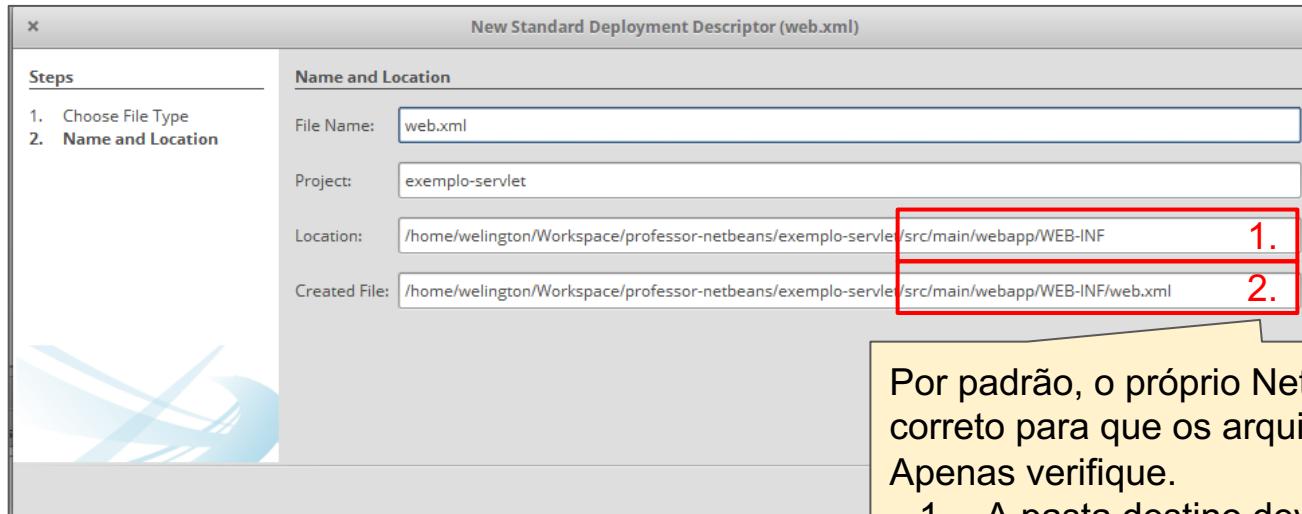
- ❑ Criando um web.xml no Netbeans.



Selecione, na escolha do arquivo, a categoria  
**(1.) Web e o tipo de arquivo (2.) Standard Deployment Descriptor (web.xml)**

# Servlets | Deployment Descriptor | Netbeans

## ❑ Criando um web.xml no Netbeans.

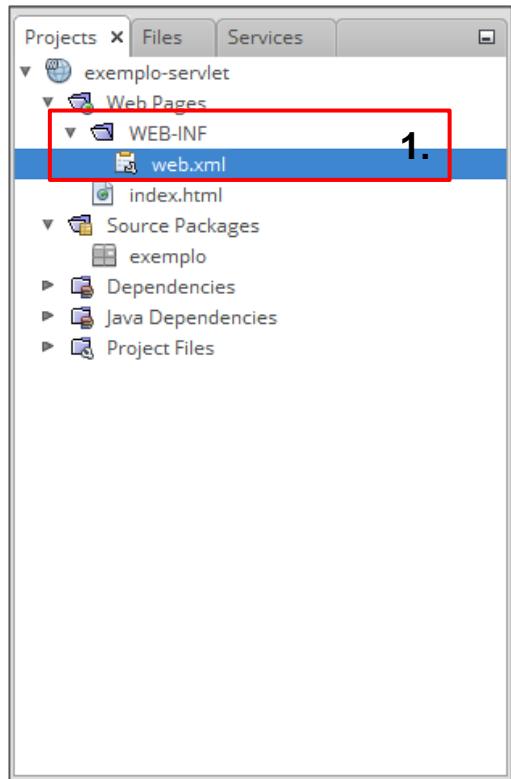


Por padrão, o próprio Netbeans irá indicar o local correto para que os arquivos sejam exibidos. Apenas verifique.

1. A pasta destino deve ser a **WEB-INF** dentro de **webapp** (para um projeto maven)
2. O arquivo deve ser **web.xml**

# Servlets | Deployment Descriptor | Netbeans

## ❑ Criando um web.xml no Netbeans.



```
web.xml

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

O arquivo gerado irá exibir apenas uma configuração, com o tempo padrão de sessão.

Vamos discutir essa configuração adiante.

# Servlets | Implementando HttpServlet

- ❑ Para implementar um HttpServlet, podemos sobrescrever métodos importantes:
  - ❑ **doGet**
    - ❑ Recebe uma requisição e uma resposta para um requisição GET.
  - ❑ **doPost**
    - ❑ Recebe uma requisição e uma resposta para uma requisição POST.
- ❑ Precisamos escrever o conteúdo dinâmico na resposta.

Contador.java

```
public class Contador extends HttpServlet {  
  
    private int count = 0;  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html;charset=UTF-8");  
        try (PrintWriter out = response.getWriter()) {  
            /* TODO output your page here. You may use following sample code. */  
            out.println("<!DOCTYPE html>");  
            out.println("<html>");  
            out.println("  <head>");  
            out.println("    <title>Servlet Contador</title>");  
            out.println("  </head>");  
            out.println("  <body>");  
            out.println("    <h1>Requisição número " + (++count) + "</h1>");  
            out.println("  </body>");  
            out.println("</html>");  
        }  
    }  
}
```

# Servlets | Implementando HttpServlet

web.xml

```
public class Contador extends HttpServlet {  
  
    private int count = 0;  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html;charset=UTF-8");  
        try (PrintWriter out = response.getWriter()) {  
            /* TODO output your page here. You may use following sample code. */  
            out.println("<!DOCTYPE html>");  
            out.println("<html>");  
            out.println("  <head>");  
            out.println("    <title>Servlet Contador</title>");  
            out.println("  </head>");  
            out.println("  <body>");  
            out.println("    <h1>Requisição número " + (++count) + "</h1>");  
            out.println("  </body>");  
            out.println("</html>");  
        }  
    }  
}
```

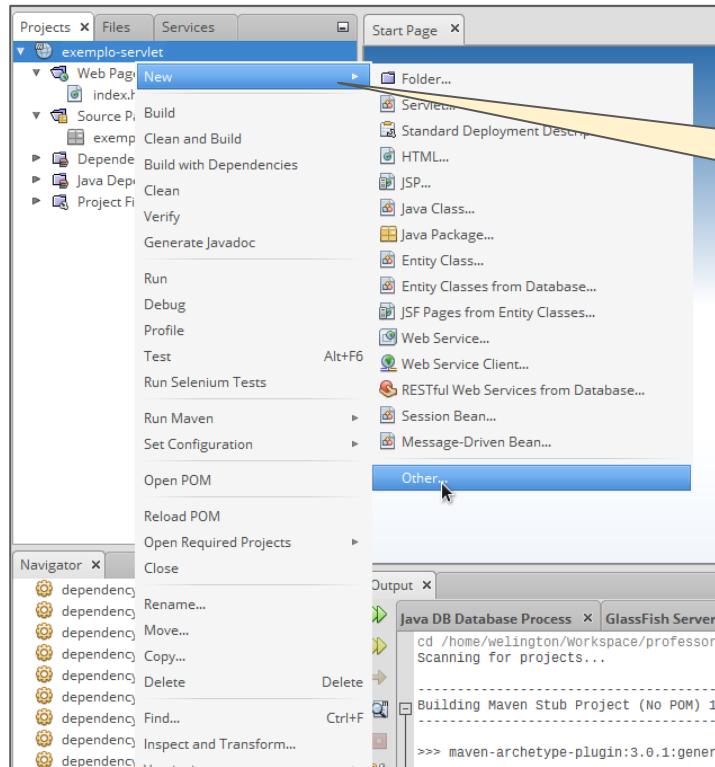
Precisamos herdar de **HttpServlet**.

O conteúdo, status code e cabeçalhos da resposta pode ser escrito no objeto **response**.

Sempre que recebermos uma requisição **GET** o código do método **doGet** será executado.

# Servlets | Implementando HTTPServlet | Netbeans

## □ Criando um Servlet no Netbeans.

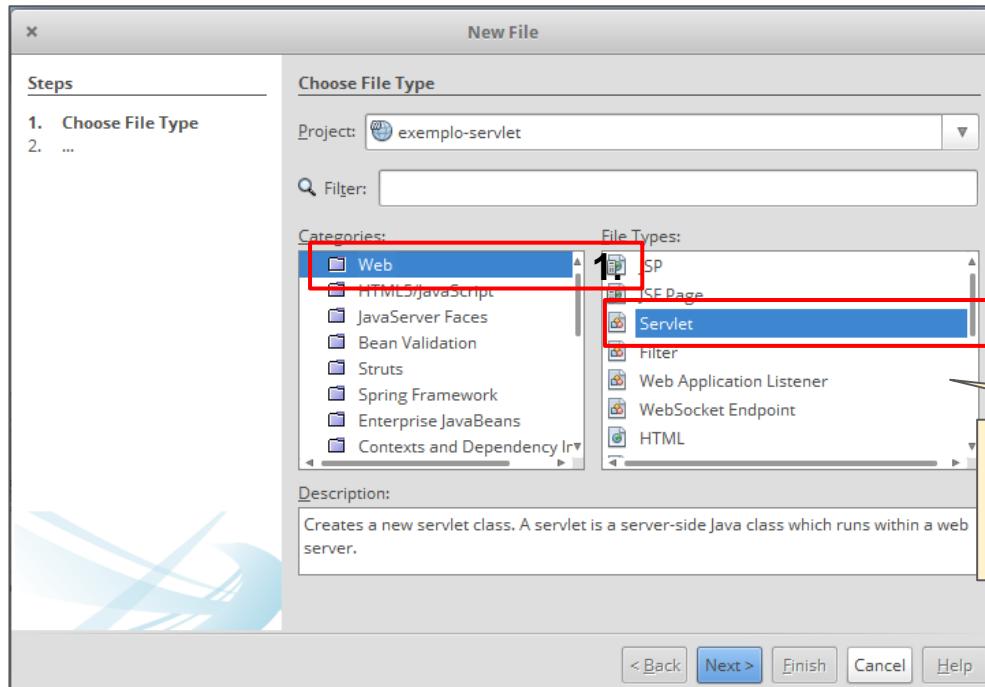


Clique com o botão direito no nome do projeto.

Selecione **New > Other...**

# Servlets | Implementando HTTPServlet | Netbeans

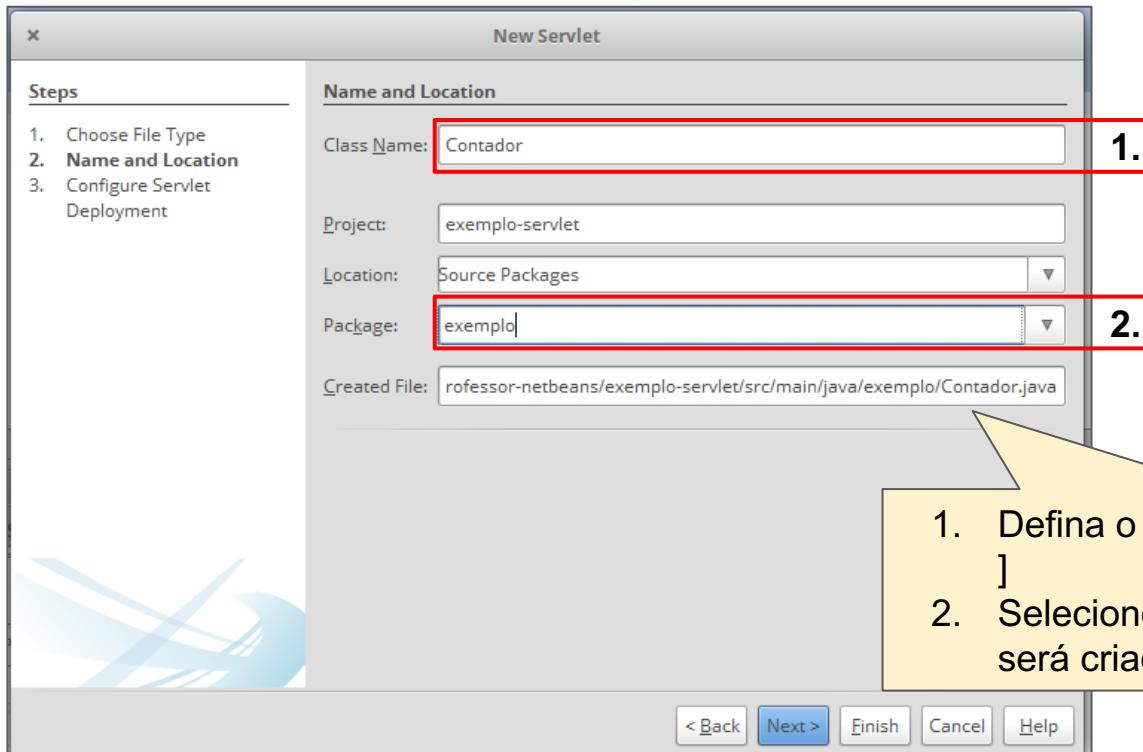
## ❑ Criando um Servlet no Netbeans.



Selezione, na escolha do arquivo, a categoria **(1.) Web** e o tipo de arquivo **(2.) Servlet**

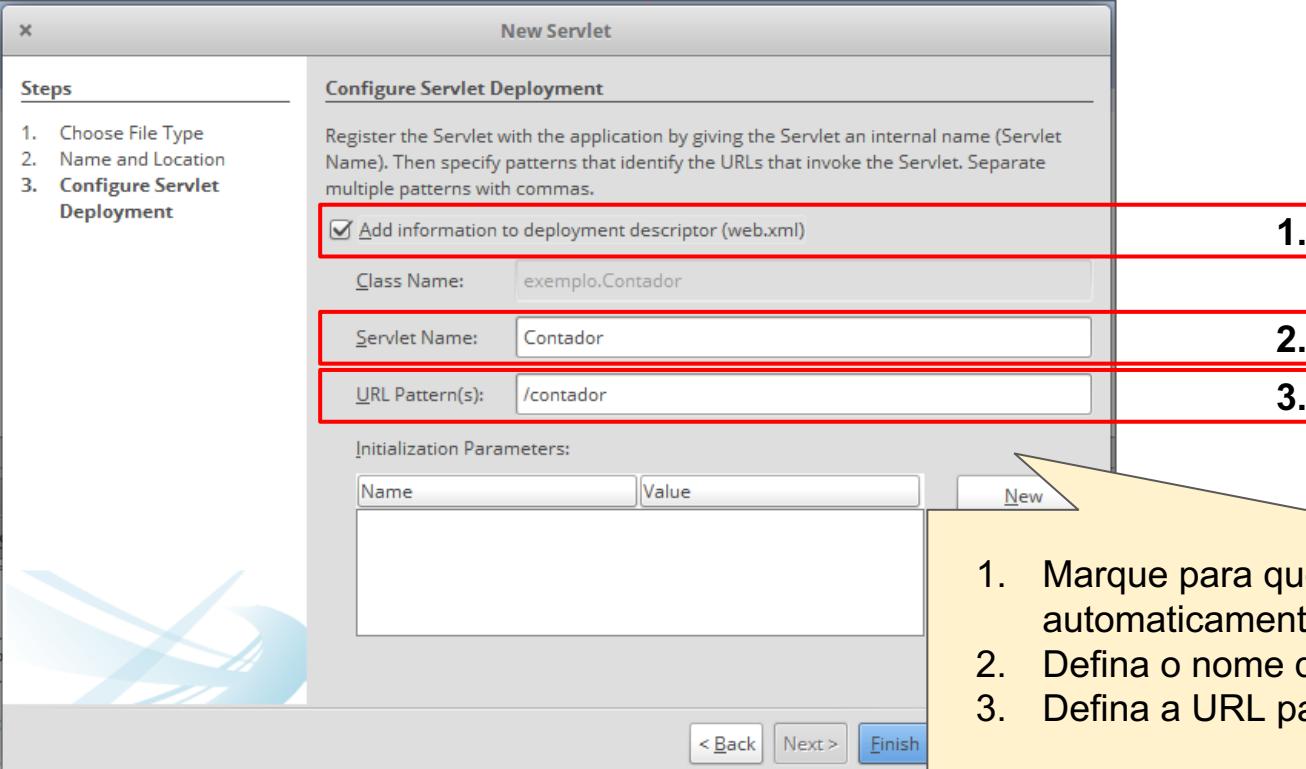
# Servlets | Implementando HTTPServlet | Netbeans

## ❑ Criando um Servlet no Netbeans.



# Servlets | Implementando HTTPServlet | Netbeans

## ❑ Criando um Servlet no Netbeans.



The screenshot shows the 'New Servlet' wizard in Netbeans. The 'Configure Servlet Deployment' step is selected. A red box highlights the first three configuration fields:

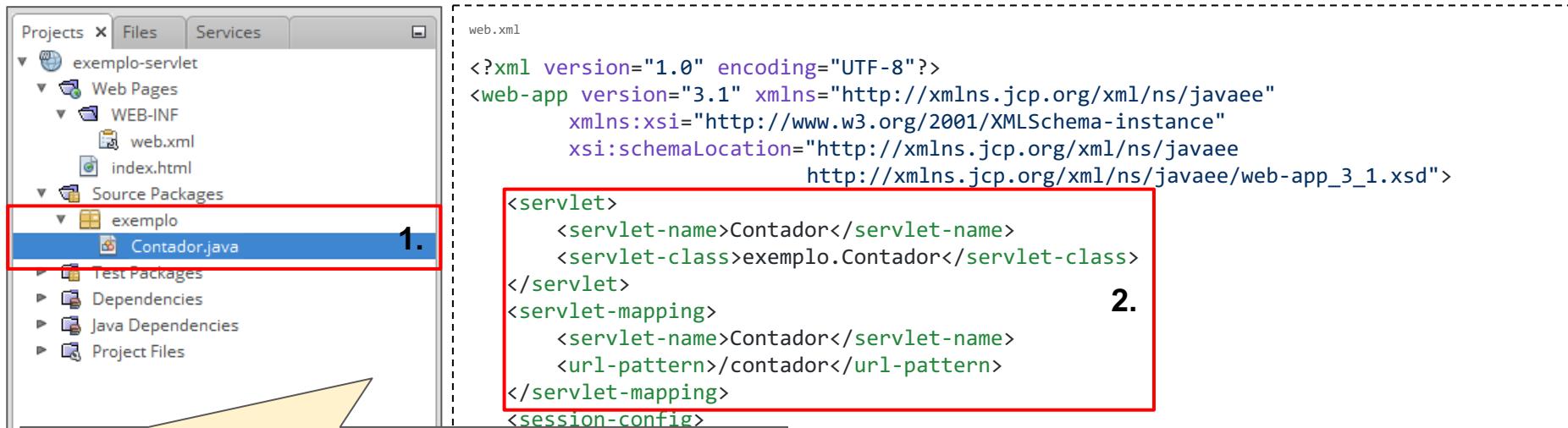
- 1.**  Add information to deployment descriptor (web.xml)
- 2.** Servlet Name: Contador
- 3.** URL Pattern(s): /contador

A yellow callout box on the right lists the steps:

1. Marque para que o servlet seja adicionado automaticamente ao **web.xml**.
2. Defina o nome do Servlet.
3. Defina a URL para acessar ao servlet.

# Servlets | Implementando HTTPServlet | Netbeans

## □ Criando um Servlet no Netbeans.



The screenshot shows the Netbeans IDE interface. On the left, the Project Explorer window displays a project named "exemplo-servlet". Inside, there are "Web Pages" and "Source Packages". Under "Source Packages", there is a package named "exemplo" containing a file named "Contador.java". A red box labeled "1." highlights this file. To the right, the "Files" tab is selected, showing the "web.xml" file. The XML content defines a servlet named "Contador" with the class "exemplo.Contador" and maps it to the URL pattern "/contador". A red box labeled "2." highlights this section of the XML code.

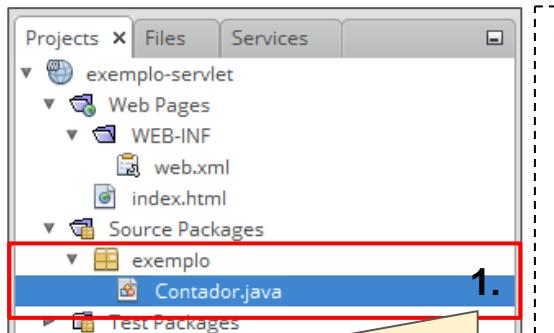
```
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

    <servlet>
        <servlet-name>Contador</servlet-name>
        <servlet-class>exemplo.Contador</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Contador</servlet-name>
        <url-pattern>/contador</url-pattern>
    </servlet-mapping>
    <session-config>
```

1. A classe que implementa o Servlet será criada no pacote de código fonte.
2. O mapeamento, conforme definido, será adicionado automaticamente ao **web.xml**.

# Servlets | Implementando HTTPServlet | Netbeans

## □ Criando um Servlet no Netbeans.



1. A classe que implementa o Servlet será criada no pacote de código fonte.
2. Os métodos **doGet** e **doPost** são automaticamente implementados.
3. O método **processRequest** gerado imprime uma mensagem padrão na tela.

web.xml

```
public class Contador extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet Contador</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet Contador at " + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }

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

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

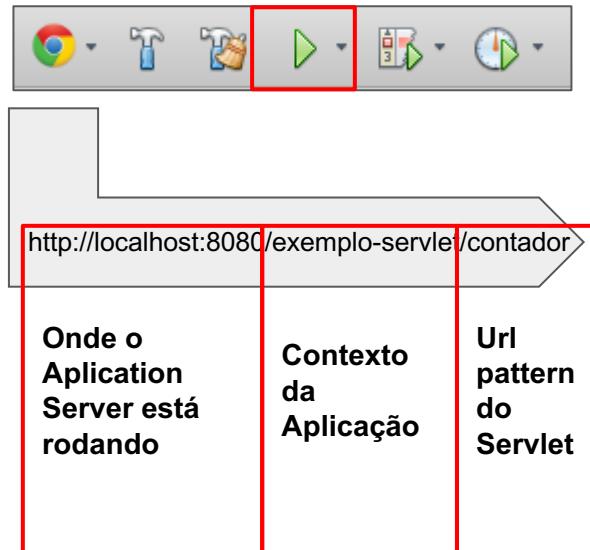
    @Override
    public String getServletInfo() {
        return "Short description";
    }
}
```

3.

2.

# Servlets | Implementando HTTPServlet | Netbeans

## ❑ Criando um Servlet no Netbeans.



# | Servlets | Implementando HTTPServlet | Netbeans



# Servlets | Implementando HTTPServlet | Netbeans

## □ Implementando o Servlet Contador.



The screenshot shows the NetBeans IDE interface. On the left, the Project Explorer window displays a Java project named "exemplo-servlet". Inside the project, there is a "Web Pages" folder containing "WEB-INF" with "web.xml" and "index.html", and a "Source Packages" folder containing a package named "exemplo" which contains the file "Contador.java". The "Contador.java" file is currently selected and shown in the code editor on the right.

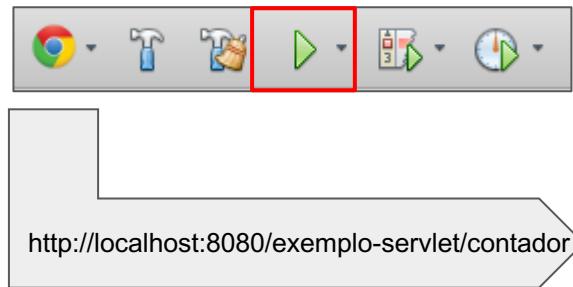
```
Contador.java
```

```
public class Contador extends HttpServlet {  
  
    private int count = 0;  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.setContentType("text/html; charset=UTF-8");  
        try (PrintWriter out = response.getWriter()) {  
            /* TODO output your page here. You may use following sample code. */  
            out.println("<!DOCTYPE html>");  
            out.println("<html>");  
            out.println("  <head>");  
            out.println("    <title>Servlet Contador</title>");  
            out.println("  </head>");  
            out.println("  <body>");  
            out.println("    <h1>Requisição número " + (++count) + "</h1>");  
            out.println("  </body>");  
            out.println("</html>");  
        }  
    }  
}
```

API completa: <http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>

# Servlets | Implementando HTTPServlet | Netbeans

## ☐ Implementando o Servlet Contador.



# Servlets | Implementando HTTPServlet | Netbeans

## ❑ Requisição/Resposta no Servlet Contador.



### Requisição

```
GET /exemplo-servlet/contador HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/59.0.3071.115 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: pt,en-US;q=0.8,en;q=0.6
```

```
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition
4.1.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish
Server Open Source Edition 4.1.1 Java/Oracle
Corporation/1.8)
Content-Type: text/html;charset=UTF-8
Date: Sun, 13 Aug 2017 22:22:17 GMT
Content-Length: 151
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Servlet Contador</title>
  </head>
  <body>
    <h1>Requisição número 17</h1>
  </body>
</html>
```

### Resposta



# Servlets | HttpServletRequest | Principais métodos

- ❑ **getContextPath()**
  - ❑ Parte da URL que corresponde à requisição realizada.
- ❑ **getHeader(String name)**
  - ❑ Obtém um cabeçalho pelo Nome do mesmo.
- ❑ **getHeaderNames()**
  - ❑ Retorna um enumerado com os nomes dos cabeçalhos.
- ❑ **getMethod()**
  - ❑ Retorno o método HTTP da requisição.
- ❑ **getRequestURI()**
  - ❑ Informações do caminho enviados pelo cliente na requisição.
- ❑ **getQueryString()**
  - ❑ Parâmetros de consulta enviados na requisição.

Lendo o corpo de uma requisição POST

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

    StringBuilder sb = new StringBuilder();
    String line = null;
    try {
        BufferedReader reader = request.getReader();
        while ((line = reader.readLine()) != null)
            sb.append(line);
    } catch (Exception e) {
        /*report an error*/
    }
    System.out.println(sb.toString());
}
```

# Servlets | HttpServletResponce | Principais métodos

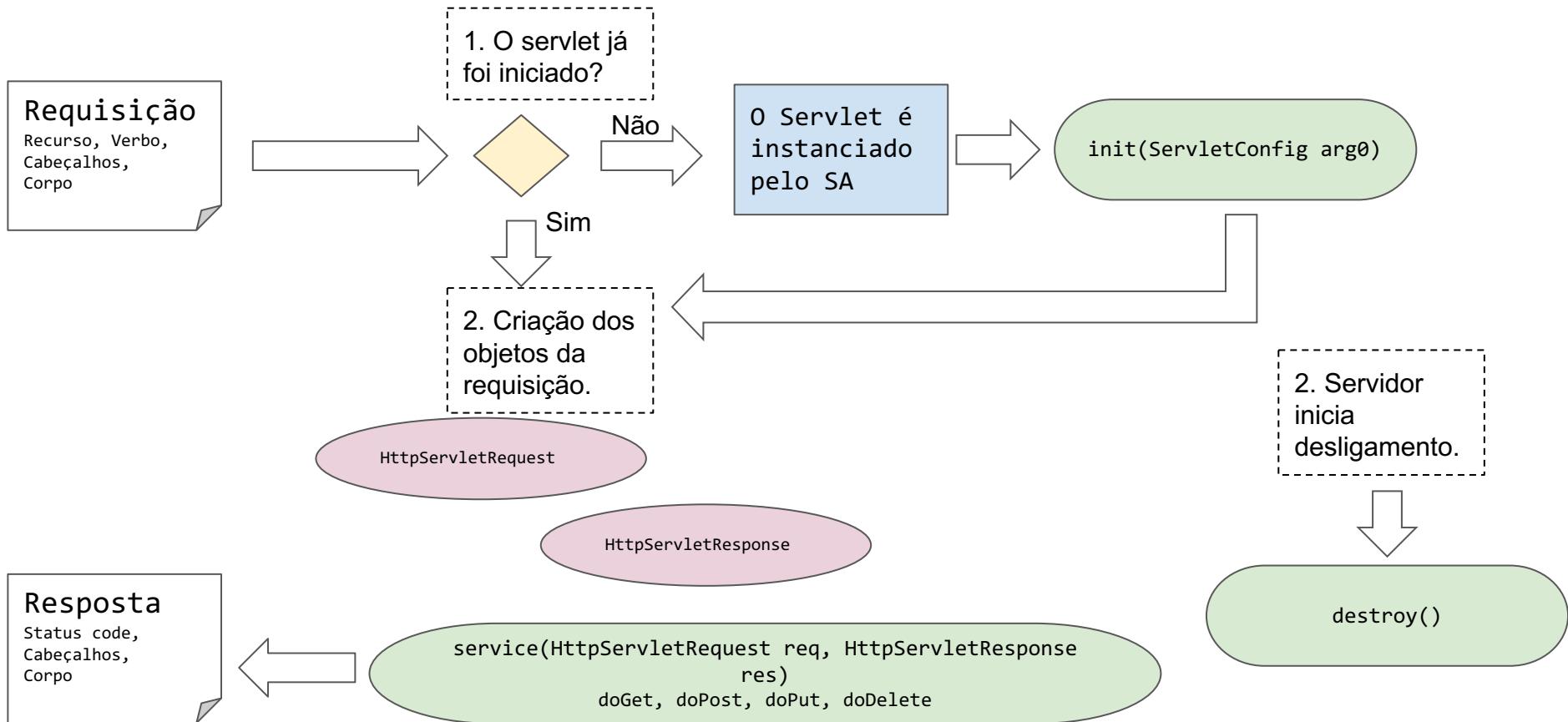
- ❑ **addHeader(String name, String value)**
  - ❑ Adiciona um cabeçalho à resposta.
- ❑ **setHeader(String name, String value)**
  - ❑ Define o valor de um cabeçalho.
- ❑ **setStatus(int sc)**
  - ❑ Define o código de status da resposta.
- ❑ **sendError(int sc)**
  - ❑ Limpa o corpo da mensagem e envia um código de erro para o cliente.
- ❑ **sendRedirect(String location)**
  - ❑ Redireciona a requisição para outro recurso.
- ❑ **setContentType(String type)**
  - ❑ Define o conteúdo da resposta.

Escrevendo o corpo de uma resposta GET/POST

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

    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code.
 */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title>Servlet Contador</title>");
        out.println("  </head>");
        out.println("  <body>");
        out.println("    <h1>Requisição número " + (++count) +
"</h1>");
        out.println("  </body>");
        out.println("</html>");
    }
}
```

# Servlets | Ciclo de Vida



# Servlets | Importância do ciclo de vida

- ❑ Alguns servlets podem ter regras complexas para serem iniciados:
  - ❑ Criar um diretório, estabelecer conexões, inicializar serviços, carregar recursos na memória, etc.
- ❑ É importante liberar recursos adicionais inicializados com o Servlet.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <servlet>
        <servlet-name>olamundo</servlet-name>
        <servlet-
class>com.exemplo.controller.OlaMundo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>olamundo</servlet-name>
        <url-pattern>/ola-mundo</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

# Servlets | Importância do ciclo de vida

- ❑ Alguns servlets podem ter regras complexas para serem iniciados:
  - ❑ Criar um diretório, estabelecer conexões, inicializar serviços, carregar recursos na memória, etc.
- ❑ É importante liberar recursos adicionais inicializados com o Servlet.



# Servlets | Importância do ciclo de vida

- ❑ Crie o Servlet **CicloDeVidaServlet** mapeado para a URL **/ciclo-vida**.
- ❑ Nesse Servlet sobrescreva os métodos **init**, **service** e **destroy** conforme exemplo ao lado.
- ❑ Realize testes e verifique o ciclo de vida.

CicloDeVida.java

```
public class CicloDeVida extends HttpServlet {  
  
    @Override  
    public void init(ServletConfig conf) throws ServletException {  
        System.out.println("Método init("+conf+");");  
    }  
  
    @Override  
    public void service(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        System.out.println("Método service("+req+","+res+");");  
    }  
  
    @Override  
    public void destroy() {  
        System.out.println("Método destroy());");  
    }  
}
```

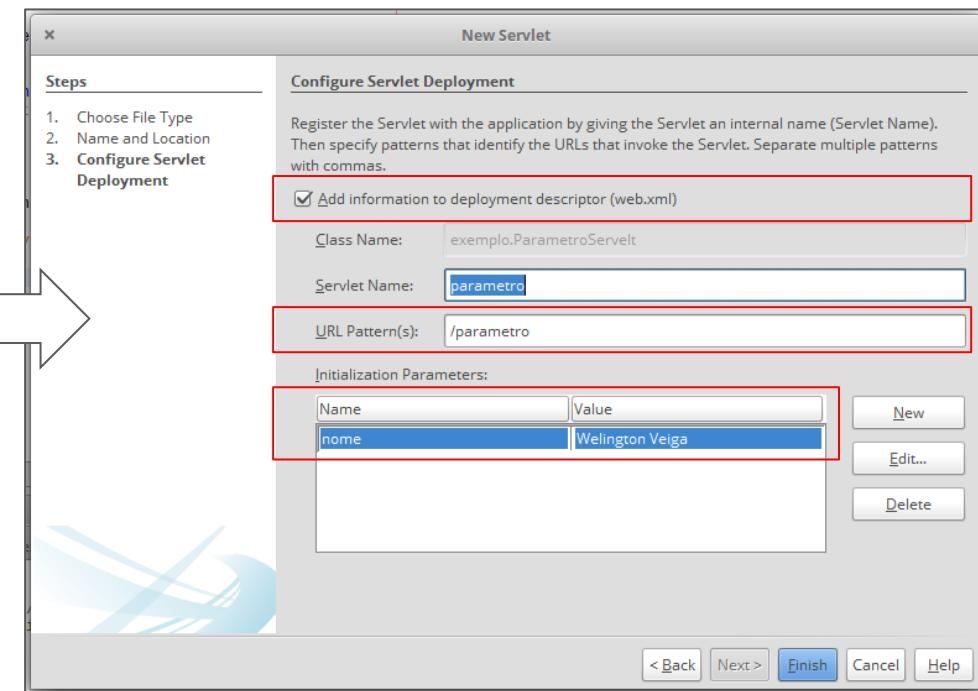
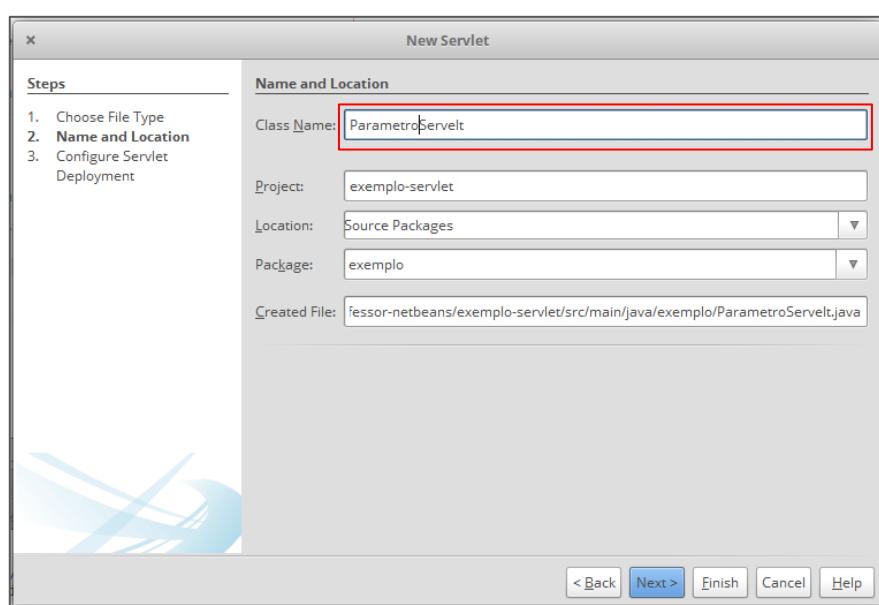
Executado quando o Servlet é acionado a primeira vez.

Executado a cada requisição.

Executado quando o Servlet é finalizado.

# Servlets | Parâmetros de Inicialização

- Servlets suportam parâmetros de inicialização, que permitem a configuração do comportamento do servlet.



# Servlets | Parâmetros de Inicialização

- Os parâmetros são definidos no `web.xml`.

`web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
    <servlet>
        <servlet-name>parametro</servlet-name>
        <servlet-class>exemplo.ParametroServlet</servlet-class>
        <init-param>
            <param-name>nome</param-name>
            <param-value>Tassio Sirqueira</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>parametro</servlet-name>
        <url-pattern>/parametro</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Nome e valor

# Servlets | Parâmetros de Inicialização

- ❑ Os parâmetros são definidos no **web.xml**.
- ❑ Podem ser obtidos no Servlet por meio do **ServletConfig**.
  - ❑ Disponível no método **init**.
  - ❑ Acessível via **getServletConfig()**

```

public class ParametroServlet extends HttpServlet {
    @Override
    public void init(ServletConfig config) throws ServletException {
        System.out.println("Nome padrão:" + config.getInitParameter("nome"));
    }

    @Override
    public void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setCharacterEncoding("UTF-8");

        String nome = request.getParameter("nome");
        if (nome == null) {                                         Aqui podemos verificar o valor padrão configurado.
            nome = getServletConfig().getInitParameter("nome");   A qualquer momento
        }                                                       por meio do método
                                                               getServletConfig()

        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("  <head>");
            out.println("    </head>");
            out.println("  <body>");
            out.println("    <h1>Nome: " + nome + "</h1>");
            out.println("  </body>");
            out.println("</html>");
        }
    }
}
  
```

# ■ Servlets | Servlet 3.0 e anotações

- ❑ Vimos que o deployment descriptor (**web.xml**) informa ao Servidor de Aplicação quais classes são Servlets e aos recursos (URLs) aos quais eles correspondem.
- ❑ A partir da versão 3.0, existe uma forma mais simples de criar Servlets.
- ❑ Por meio de anotações.
  - ❑ Recurso inserido na versão 6 do Java.
  - ❑ Permite “anotar” classes com informações adicionais de configuração.



# Servlets | Servlet 3.0 e anotações

- ❑ Mantenha o web.xml para configurações adicionais da aplicação.
- ❑ O arquivo ainda é obrigatório.

```
web.xml

<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```



# Servlets | Servlet 3.0 e anotações

- ❑ Crie Servlets utilizando apenas a anotação  
**@WebServlet**

web.xml

```
@WebServlet(value="/olamundo", name="ServletOiMundo")
public class OiMundo extends HttpServlet {
    protected void service(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
        String nome = request.getParameter("nome");
        PrintWriter out = response.getWriter();
        out.println("Ola Mundo Servlet 3: " + nome);
    }
}
```



# | Servlets | Servlet 3.0 e anotações

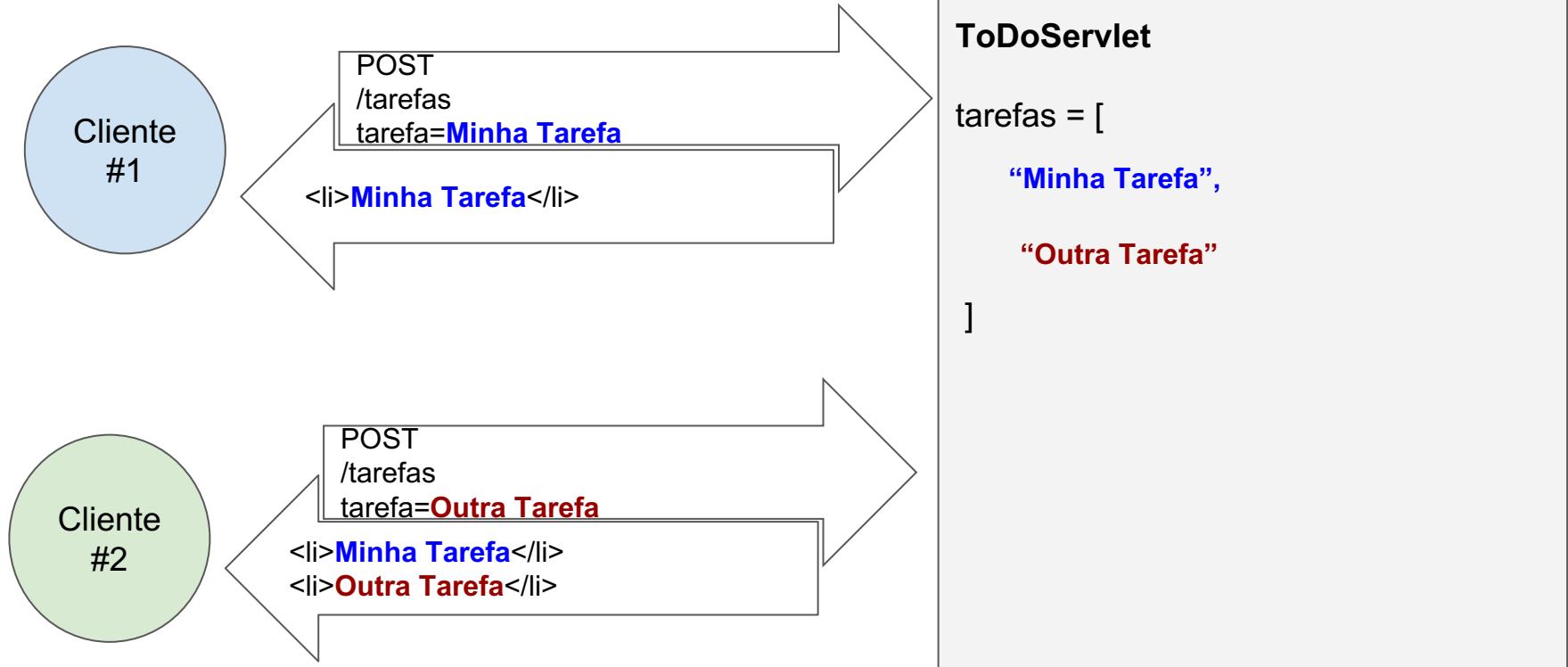


# ■ Servlets | Servlet 3.0 e anotações

- ❑ **Atenção:** não misture no mesmo projeto anotações com XML. Mesmo funcionando, prejudica a legibilidade do código do projeto,



# Servlets | Armazenando Informações | Problema



# ■ Servlets | Armazenando Informações | Problema

- ❑ Informações relacionadas à um cliente específico **não podem** ser armazenadas como atributo no Servlet.
- ❑ Todas as requisições compartilham a **mesma instância de Servlet**.



# Servlets | Armazenando Informações | Problema

- ❑ Informações relacionadas à um cliente específico **não podem** ser armazenadas como atributo no Servlet.
- ❑ Todas as requisições compartilham a **mesma instância de Servlet**.
- ❑ Como o **HTTP não mantém estado** de sessão, como resolver esse problema?
- ❑ Exemplos:
  - ❑ Acesso e login
  - ❑ Carrinho de compras.
  - ❑ Preferências



# Servlets | Armazenando Informações | Problema

- ❑ Como o protocolo não suporta sessões, as aplicações precisam fazer esse controle.
- ❑ Dessa forma é necessário uma forma de identificar um cliente específico.
- ❑ Para isso existem duas estratégias principais:
  - ❑ Cookies.
  - ❑ Parâmetro adicional na URL.



# Servlets | Sessões

- ❑ O JavaEE oferece uma abstração para tratar dados de sessão.
  - ❑ A interface **HttpSession**
- ❑ Podemos criar ou obter a sessão existente a partir de uma requisição.

Servlet.java

```
HttpSession session = request.getSession(true);  
  
if (session.isNew()) {  
  
    session.setAttribute("corPreferida", "#000000");  
  
}  
  
}
```

Obtém a sessão existente ou cria uma nova sessão para aquele cliente.

O **isNew** verifica se a sessão acabou de ser criada ou já existia.

Adicionamos atributos na sessão por meio do **setAttribute**.

```
String cor = (String) session.getAttribute("corPreferida");
```

Obtemos atributos na sessão por meio do **getAttribute**.

O cliente **não** tem acesso a esses dados que ficam apenas no servidor.

# Servlets | Sessões | Cookies

- ❑ Como o servidor reconhece um cliente para carregar a sessão correta?
  - ❑ Por meio do **JSESSIONID**, uma identificação randômica gerada para cada sessão nova.
  - ❑ Essa sessão pode ser armazenada em um cookie ou enviada como parâmetro na URL.
- ❑ Onde os dados da sessão são armazenados?
  - ❑ O Servidor de Aplicação pode armazenar na memória, no disco ou em um banco de dados, por exemplo.
  - ❑ Todo objeto na sessão deve ser **serializável**.



# Servlets | Sessões | Cookies

## ❑ Mas o que são **cookies**?

- ❑ Um Cookie HTTP são pequenas porções de informações que o servidor pode enviar para o Navegador.
- ❑ O navegador armazena essas informações e devolve para o servidor quando faz novas requisições.
- ❑ Assim é possível guardar preferências, sessões, carrinhos de compras, histórico de buscas entre outras informações pequenas.
- ❑ O navegador pode não suportar cookies e o usuário pode desabilitá-los.

## ❑ Cookies **não são seguros**, podem ser lidos no lado cliente, por isso não devem conter Informações sensíveis.



# Servlets | Sessões | Cookies

## ❑ Como funciona?



```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco;tasty_cookie=strawberry
```

# Servlets | Sessões | Cookies

- ❑ Mas e se os **cookies** estiverem desabilitados?
  - ❑ Um parâmetro é adicionado pelo Servidor de Aplicação na URL
  - ❑ Exemplo:
    - ❑ <http://exemplo.com;JSESSIONID=udhofhoi4fojnio4no4n>



# Servlets | Sessões | Cookies | Exemplo

```
public class Contador extends HttpServlet {  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
  
        HttpSession session = request.getSession(true);  
        if (session.isNew()) {  
            session.setAttribute("count", 0);  
        }  
  
        int count = (int) session.getAttribute("count");  
        count++;  
  
        session.setAttribute("count", count);  
  
        response.setContentType("text/html; charset=UTF-8");  
        try (PrintWriter out = response.getWriter()) {  
            /* TODO output your page here. You may use following sample code. */  
            out.println("<!DOCTYPE html>");  
            out.println("<html>");  
            out.println("  <head>");  
            out.println("    <title>Servlet Contador</title>");  
            out.println("  </head>");  
            out.println("  <body>");  
            out.println("    <h1>Requisição número " + count + "</h1>");  
            out.println("  </body>");  
            out.println("</html>");  
        }  
    }  
}
```

Vamos obter uma sessão e, caso ela não exista, a mesma será criada.

Realizamos as operações necessárias e armazenamos novamente os dados como atributo da sessão.

# HTTP | Referências adicionais

1. <https://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html>
2. <https://www.caelum.com.br/apostila-java-web/servlets/>
3. <http://blog.caelum.com.br/java-ee6-comecando-com-as-servlets-3-0/>
4. <https://www.caelum.com.br/apostila-java-web/appendice-topicos-da-servlet-api/>
5. [https://www.tutorialspoint.com/servlets/servlets-form-data.htm](https://www.tutorialspoint.com/servlets/servlets_form_data.htm)

# Laboratório de Programação de Web Sites Dinâmicos

Java Server Pages

Tassio Sirqueira – 2019/01

# JSP | Por que precisamos de algo além dos Servlets?

ExemploServlet.java

```
public class ExemploServlet extends HttpServlet {

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

        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<html><head><title>Página</title><style>\n" +
"h1 {font-size: 2rem; font-weight: bolder; margin-bottom: 1rem;}\n" +
"p {margin-bottom: 1rem; color: tomato;}\n" +
"button {cursor: pointer; appearance: none; border-radius: 4px; font-size: 1.25rem; padding: 0.75rem 1rem; border: 1px solid navy;}\n" + "</style></head>\n" +
" <body>\n" +
"   <h1>I am a headline made with HTML</h1>\n" +
"   <p>And I am a simple text paragraph. The color of this text is styled with CSS. Click the button below to remove me through
the" + "power JavaScript.</p>\n" +
"   if (request.getSession().getAttribute("cadastro") != null) {
        <button>Hide the text above</button>\n" +
    }
    <script>$('button').on('click', function() {\n" +
"      $('p').css('opacity', 0);\n" +
"   });</script>\n" +
" </body> \n" +
"</html>");

    }
}
```

CSS

Java

JavaScript

HTML

Java

# | JSP | Introdução



- ❑ Podemos escrever conteúdo dinâmico através de Servlets.
  - ❑ Teremos muitos problemas na manutenção das nossas páginas e também na legibilidade do nosso código
- ❑ Para melhorar a manutenabilidade de websites dinâmicos, o JavaEE oferece a tecnologia **JavaServer Pages (JSP)**.
- ❑ Essa tecnologia é baseada nos Servlets que vimos na aula anterior, portanto todos os conceitos aprendidos devem ser considerados aqui.
  - ❑ Muitas vezes vamos combinar Servlet e JSPs no desenvolvimento do aplicação web dinâmicas.

# JSP | Introdução



## ❑ Exemplo de JSP:

bemvindo.jsp

```
<%@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>
        <h1>Hello World!</h1>
    </body>
</html>
```

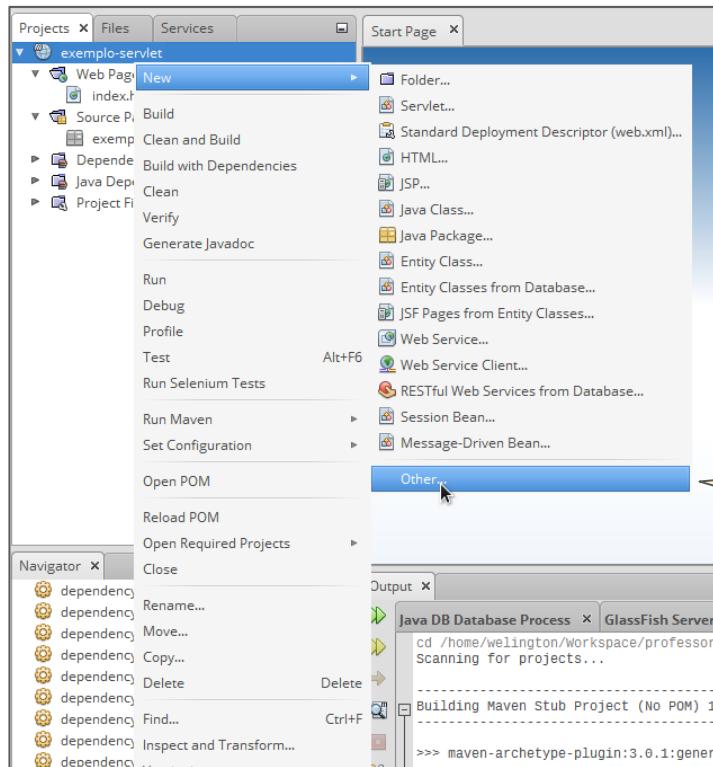
Podemos adicionar diretivas especiais por meio de tags no formato <%@ .... %>

Podemos escrever HTML normal em JSPs



# JSP | Criando JSPs no Netbeans

## ☐ Criando um JSP:



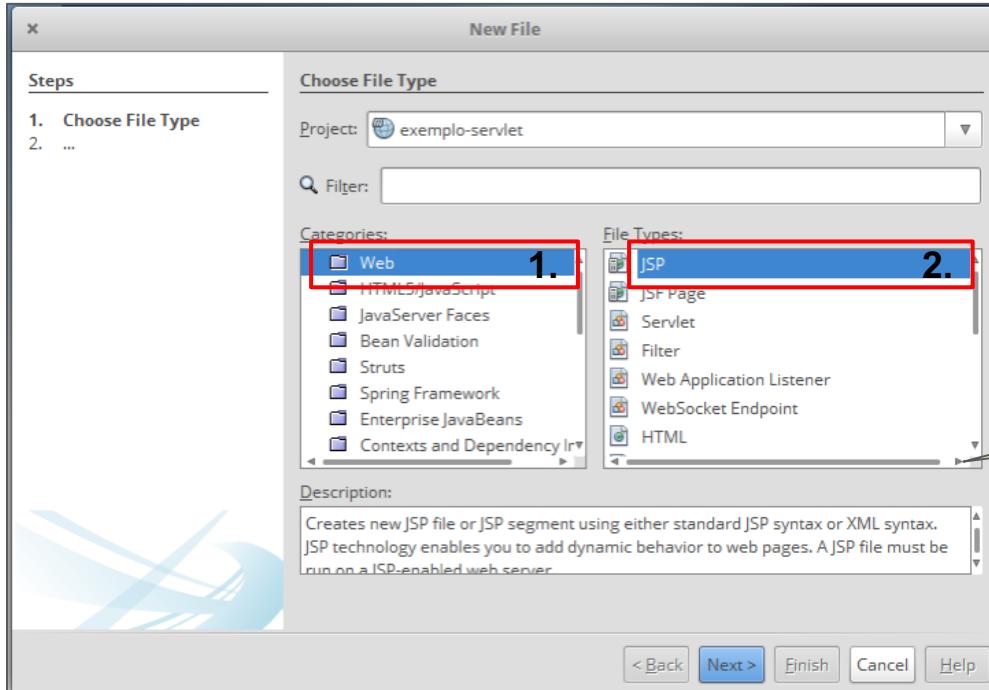
Clique com o botão direito no nome do projeto.

Selecione **New > Other...**

# JSP | Criando JSPs no Netbeans



## Criando um JSP:

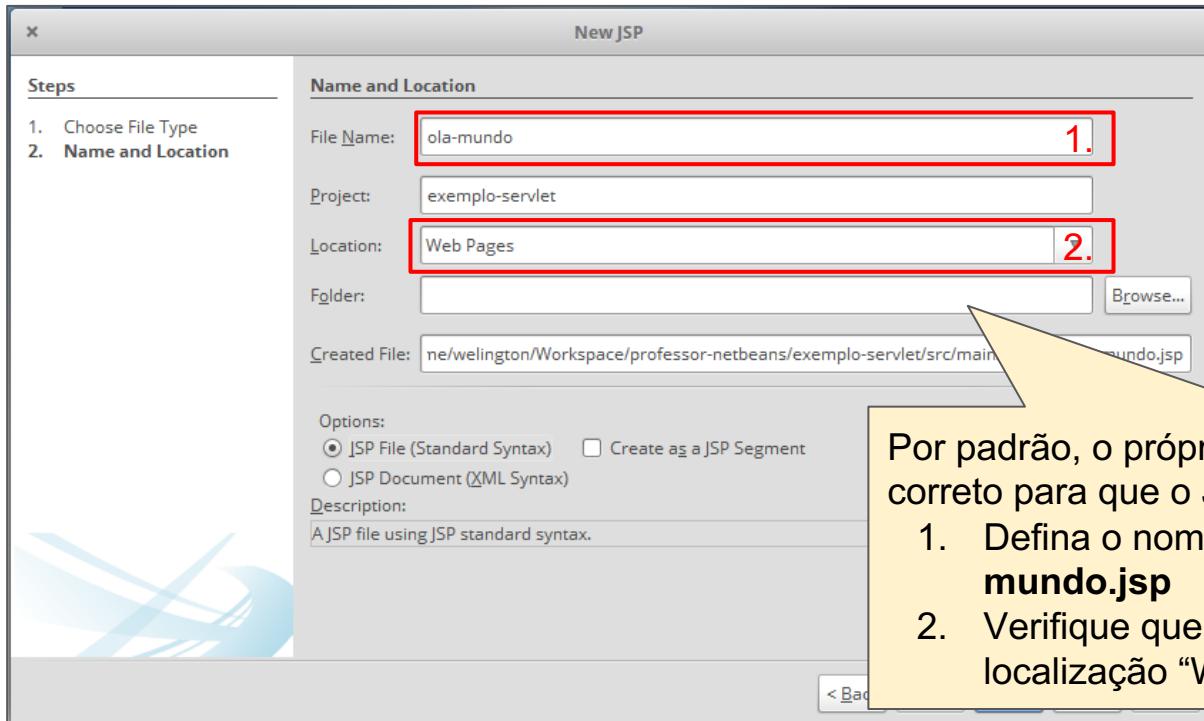


Selecione, na escolha do arquivo, a categoria  
**(1.) Web** e o tipo de arquivo **(2.) JSP**



# JSP | Criando JSPs no Netbeans

## ❑ Criando um JSP:



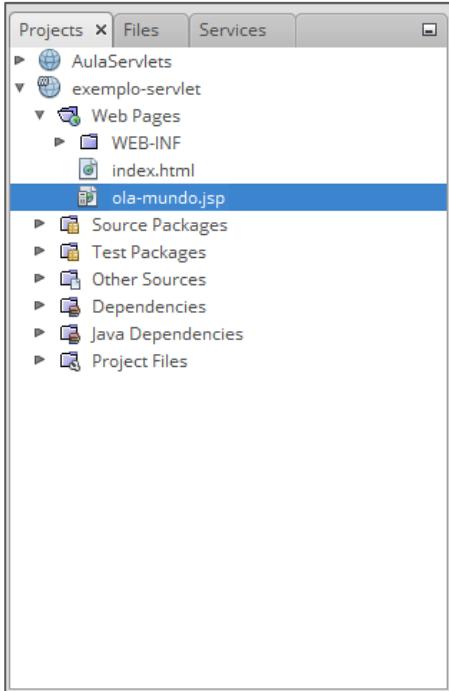
Por padrão, o próprio Netbeans irá indicar o local correto para que o JSP criados.

1. Defina o nome da página JSP. No exemplo **ola-mundo.jsp**
2. Verifique que o mesmo deve ser criado na localização “Web Pages”

# JSP | Criando JSPs no Netbeans



## Criando um JSP:



```
ola-mundo.jsp

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

<!DOCTYPE html>

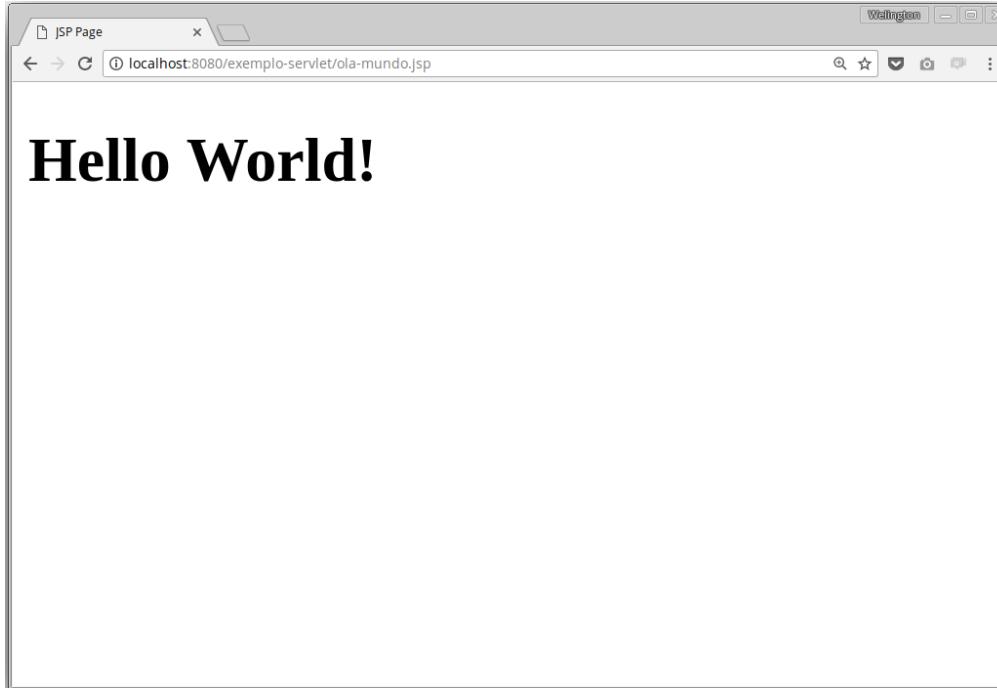
<html>

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

# | JSP | Criando JSPs no Netbeans



- ❑ Acessando o JSP criado:
  - ❑ <http://localhost:8080/exemplo-servlet/ola-mundo.jsp>



# | JSP | Scriptlet

- ❑ Embora suporte HTML estático, a vantagem de se usar JSPs consiste na possibilidade de adicionar **elementos dinâmicos** às páginas.
- ❑ A forma mais simples de se fazer isso é por meio de **Scriptlets**.
  - ❑ Pequenas porções de lógica de negócio resolvidas na página.
  - ❑ Escritas em Java.
- ❑ Podemos escrever scriptlets por meio da sintaxe `<% ... %>`





# JSP | Scriptlet

## ❑ Exemplo de scriptlet

bemvindo.jsp

```
<%@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>
        <h1><% out.println("Olá Mundo!");%></h1>
    </body>
</html>
```

Podemos escrever código Java em qualquer parte da página.



# JSP | Scriptlet

## ❑ Exemplo de scriptlet

ola-mundo.jsp

```
<%@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>
        <%
            String nome = "Tassio Sirqueira";
        %>
        <h1><% out.println("Visitante: "+ nome);%></h1>
    </body>
</html>
```

Podemos escrever código Java em qualquer parte da página.

# | JSP | Scriptlet | Objetos Implícitos

- ❑ Existem algumas variáveis especiais disponíveis no escopo de uma página JSP.
- ❑ São conhecidos como **Objetos Implícitos**.
- ❑ Os principais Objetos Implícitos são familiares:
  - ❑ **out**: OutputStreamer da resposta atual.
  - ❑ **request**: Requisição atual.
  - ❑ **response**: Resposta atual.
  - ❑ **session**: Sessão de usuário atual.





# JSP | Scriptlet | Objetos Implícitos

## Exemplo

```
contador.jsp
<%@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>
        <%
            Integer count = (Integer) session.getAttribute("count");
            if (count == null) {
                count = 0;
            }
            count++;
            session.setAttribute("count", count);
        %>
        <h1>Você visitou essa página <% out.println(count);%> vezes</h1>
    </body>
</html>
```

Podemos re-implementar o contador usando apenas objetos implícitos do JSP.

# | JSP | Scriptlet | Sintaxes especiais

- ❑ Além da sintaxe <% ... %>, existem outras sintaxes bastante úteis para a construção de páginas dinâmicas.
- ❑ Para declarar métodos e variáveis no JSP: <%! ... %>.

```
<%
    int count;
    private int incrementa() {
        return ++count;
    }
%>
```

- ❑ Para imprimir sem necessidade de **out.println** <%= ... %>.

```
<%= incrementa() %>
```

# | JSP | Scriptlet | Sintaxes especiais

- ❑ Para fazer comentários que não farão parte da página:

`<%-- ... --%>.`

```
<%-- This is JSP comment --%>
```

- ❑ Para importar bibliotecas : `<%@page import="..." %>`.

```
<%@page import="java.util.Date"%>
```

# | JSP | Scriptlet | Controle de Fluxo

- ❑ Podemos adicionar instruções if...else

```
<%! int day = 3; %>
<html>
    <head><title>IF...ELSE Example</title></head>

    <body>
        <% if (day == 1 | day == 7) { %>
            <p> Today is weekend</p>
        <% } else { %>
            <p> Today is not weekend</p>
        <% } %>
    </body>
</html>
```

Podemos alternar instruções Java com HTML  
ou usar **out.println**.

# | JSP | Scriptlet | Controle de Fluxo

- ❑ Podemos adicionar instruções switch...case

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
    <% switch(day) {
        case 1:
            out.println("It\'s Monday.");
            break;
        case 2:
            out.println("It\'s Tuesday.");
            break;
        case 3:
            out.println("It\'s Wednesday.");
            break;
        case 4:
            out.println("It\'s Thursday.");
            break;
        case 5:
            out.println("It\'s Friday.");
            break;
    } %>
</body>
</html>
```

# | JSP | Scriptlet | Controle de Fluxo

- ❑ Podemos adicionar instruções **for, while, e do...while**

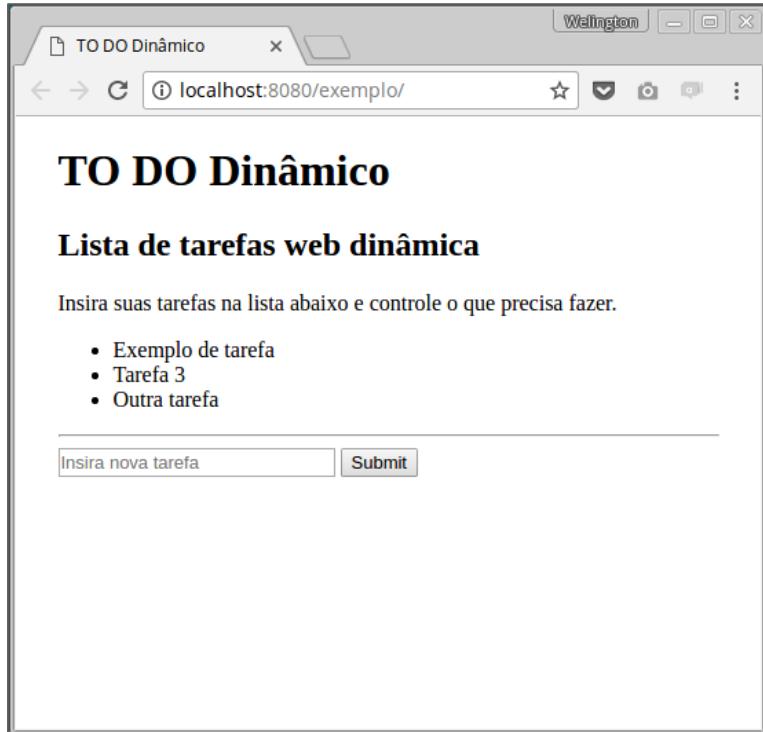
```
<%! int fontSize; %>
<html>
    <head><title>FOR LOOP Example</title></head>

    <body>
        <%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
            <font color = "green" size = "<%= fontSize %>">
                Texto
            </font><br />
        <%}%>
    </body>
</html>
```

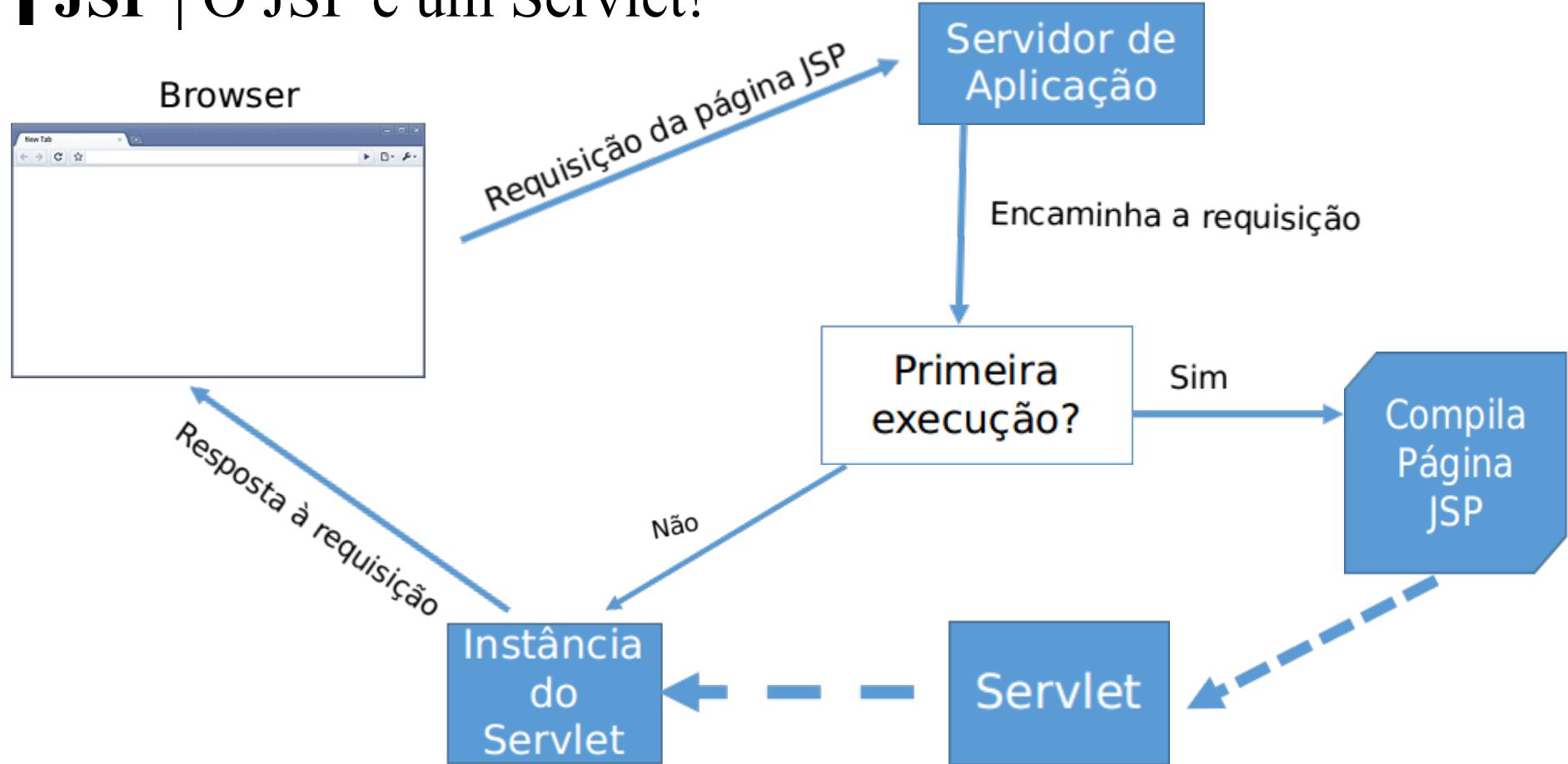
# Atividade | Implementando JSP

- ❑ Utilizando o template de TO-DO list desenvolvido na atividade 4, altere a implementação para utilizar JSP.
- ❑ Envie para tassio@tassio.eti.br com o assunto

**Atividade 6 - Todo List JSP** com seu nome no corpo do e-mail até 02/09.



# JSP | O JSP é um Servlet!



# JSP | Servlets + JSP

bemvindo.jsp

```
<%@page contentType="..." pageEncoding="..."%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="..." content="...">
    <title>JSP Page</title>
  </head>
  <body>
    <h1><% out.println("Olá Mundo!");%></h1>
  </body>
</html>
```

ola\_002dmando\_jsp.jsp

```
out.write("\n");
out.write("\n");
out.write("\n");
out.write("\n");
out.write("<!DOCTYPE html>\n");
out.write("<html>\n");
out.write("  <head>\n");
out.write("    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
out.write("    <title>JSP Page</title>\n");
out.write("  </head>\n");
out.write("  <body>\n");
out.write("    <!-- Comentário HTML --&gt;\n");
out.write("    ");
out.write("\n");
out.write("    ");
out.println("Olá Mundo!");
out.write("\n");
out.write("  &lt;/body&gt;\n");
out.write("&lt;/html&gt;\n");</pre>
```

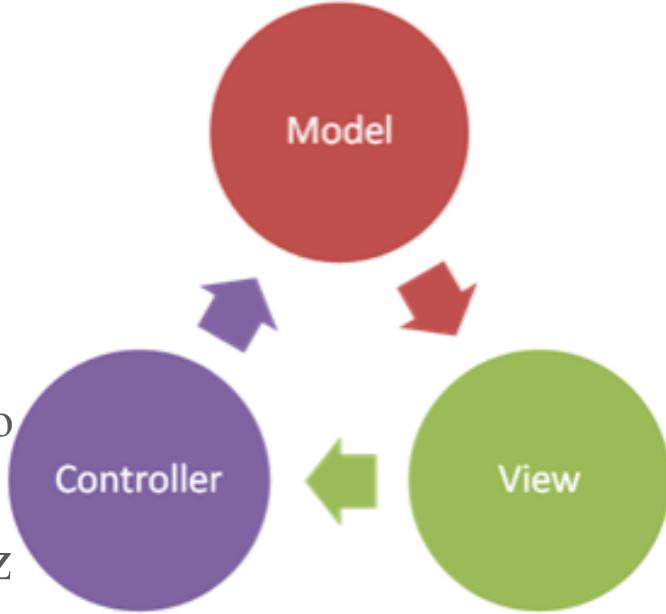
# | JSP | Servlets + JSP

- ❑ O uso de JSPs possui um inconveniente equivalente ao que vimos no Servlets.
  - ❑ No **servlet** adicionamos código **HTML** no meio do código **Java**.
  - ❑ No **JSP** adicionamos código **Java** no meio do **HTML**.
- ❑ São dois tipos diferentes de código “espaguete”.
- ❑ Mas a combinação das duas tecnologias permite uma separação mais adequada de responsabilidades.
  - ❑ Para resolver essa questão temos alguns padrões arquiteturais relevantes.
  - ❑ O mais utilizado é o padrão MVC.



# | JSP | Servlets + JSP | MVC

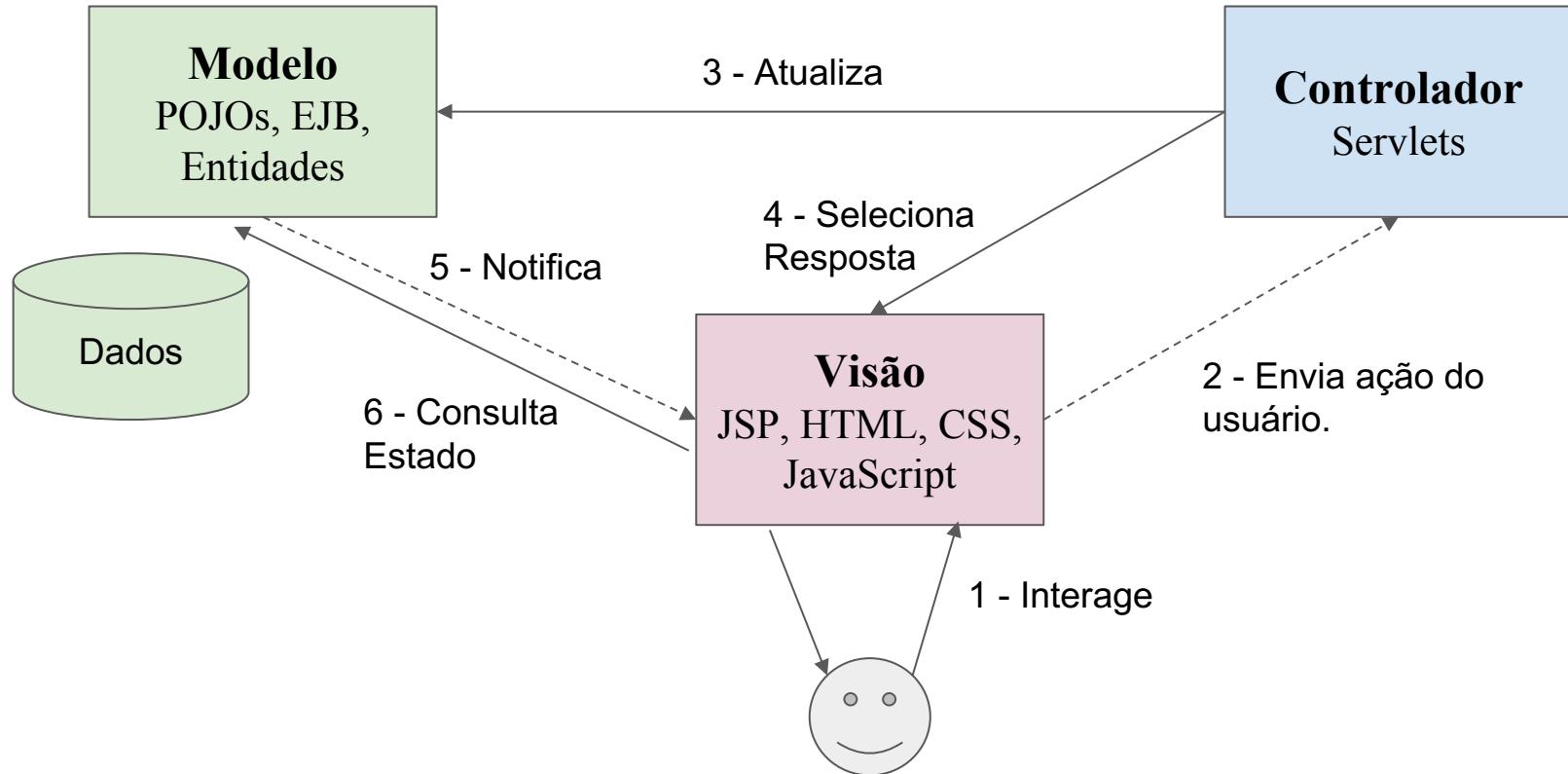
- ❑ Model View Controller é um padrão arquitetural em 3 camadas.
  - ❑ Em português: Modelo-Visão-Controlador.
  - ❑ Separa a representação da informação da interação do usuário com ela.
- ❑ O padrão MVC foi descrito pela primeira vez em 1979 por Trygve Reenskaug, que trabalhava no Smalltalk, na Xerox PARC.
- ❑ É o padrão mais utilizado na Web para a arquitetura de aplicações dinâmicas.



# | JSP | Servlets + JSP | MVC

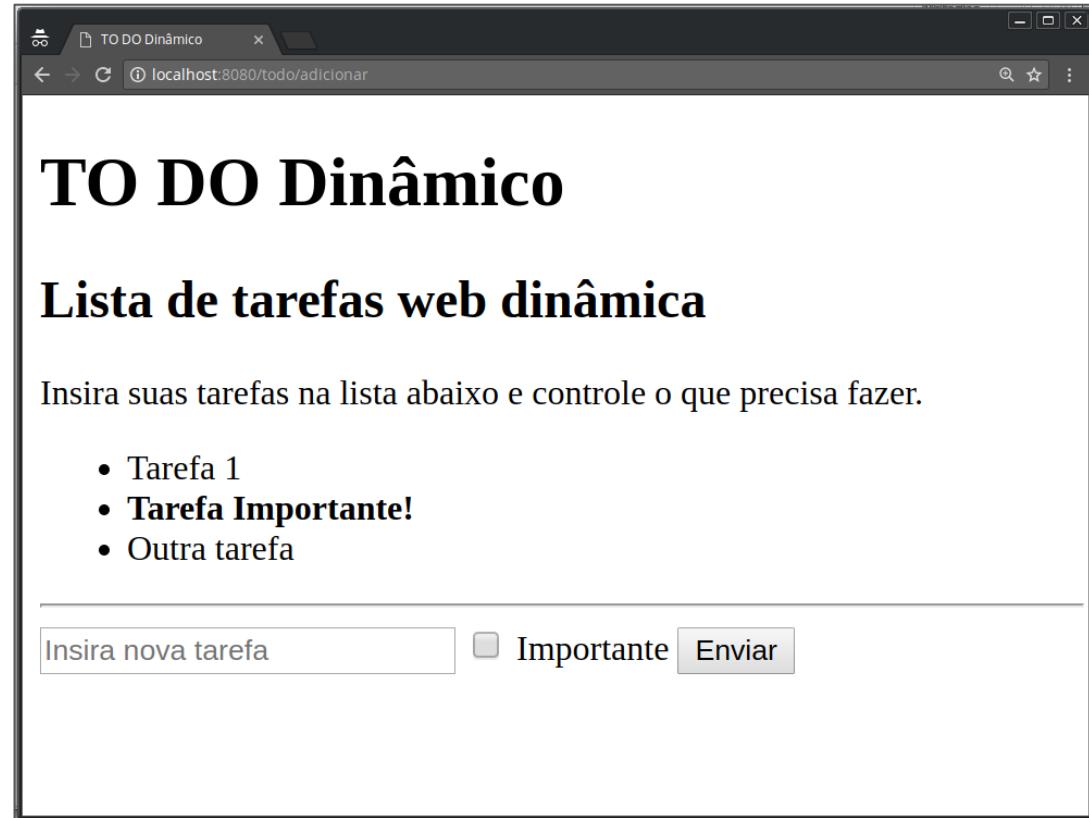
- ❑ Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles.
  - ❑ **Um controlador (controller)** envia comandos para o modelo de acordo com as ações do usuário. O controlador também pode enviar comandos para a visão associada para alterar a apresentação da visão do modelo.
  - ❑ **Um modelo (model)** armazena dados, processa as regras de negócio, e altera o estado interno da aplicação, podendo notificar as visões sobre seu estado atual.
  - ❑ **A visão (view)** Gera uma representação (Visão) dos dados presentes no modelo solicitado.

# | JSP | Servlets + JSP | MVC



# | JSP | Servlets + JSP | MVC | Modelo

- ❑ Utilizando como base o Todo List desenvolvido nas atividades anteriores, vamos desenvolver uma versão MVC.



# JSP | Servlets + JSP | MVC | Modelo

```
package todo.model;

public class Tarefa {

    private String descricao;
    private Boolean importante;

    public Tarefa(String descricao, Boolean importante) {
        this.descricao = descricao;
        this.importante = importante;
    }
    // Getters e Setters omitidos ...

    @Override
    public String toString() {
        return "Tarefa{" + "descricao=" + descricao + ", importante=" + importante + '}';
    }
}
```

Modelos podem ser  
classes simples em Java (POJOs),  
EJBs, entidades do JPA, etc.

# JSP | Servlets + JSP | MVC | Visão

```
<%@page import="java.util.List"%>
<%@page import="todo.model.Tarefa"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head> ... </head>
    <body>
        <div class="centro">
            <h1>TO DO Dinâmico</h1>
            <h2>Lista de tarefas web dinâmica </h2>
            <p>Insira suas tarefas na lista abaixo e controle o que precisa fazer.</p>
            <ul>
                <% List<Tarefa> tarefas = (List<Tarefa>) session.getAttribute("tarefas"); %>
                <% if(tarefas != null) {%
                    <% for (Tarefa tarefa : tarefas) { %
                        <% if(tarefa.getImportante()) {%
                            <li><strong><%= tarefa.getDescricao() %></strong></li>
                        <% } else {%
                            <li><%= tarefa.getDescricao() %></li>
                        <% }%
                    <% }%
                <% }%
            </ul>
            <hr />
            <form action="/todo/adicionar" method="post">
                <input type="text" name="descricao" placeholder="Insira nova tarefa" />
                <input type="checkbox" name="importante"/> Importante <input type="submit" name="enviar" value="Enviar" />
            </form>
        </div>
    </body>
</html>
```

Apenas a lógica de exibição das tarefas da sessão.

(Ainda tem muito Java aqui... podemos melhorar isso.)

# JSP | Servlets + JSP | MVC | Controlador

```
package todo.controllers;

@WebServlet(urlPatterns = "/adicionar")
public class AdicionarController extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        // 1 - Garantimos que a sessão existe.
        HttpSession session = req.getSession(true);

        // 2 - Obtemos os dados enviados e montamos o modelo Tarefa.
        String descricao = req.getParameter("descricao");
        boolean importante = "on".equalsIgnoreCase(req.getParameter("importante"));
        Tarefa tarefa = new Tarefa(descricao, importante);

        // 3 - Adicionamos a tarefa às tarefas da sessão.
        List<Tarefa> tarefas = (List<Tarefa>) session.getAttribute("tarefas");
        if (tarefas == null) {
            tarefas = new ArrayList<>();
        }
        tarefas.add(tarefa);

        // 4 - Devolvemos as tarefas para a sessão.
        session.setAttribute("tarefas", tarefas);

        // 5 - Repassamos a resposta para a View (index.jsp)
        RequestDispatcher rd = req.getRequestDispatcher("/index.jsp");
        rd.forward(req,resp);
    }
}
```

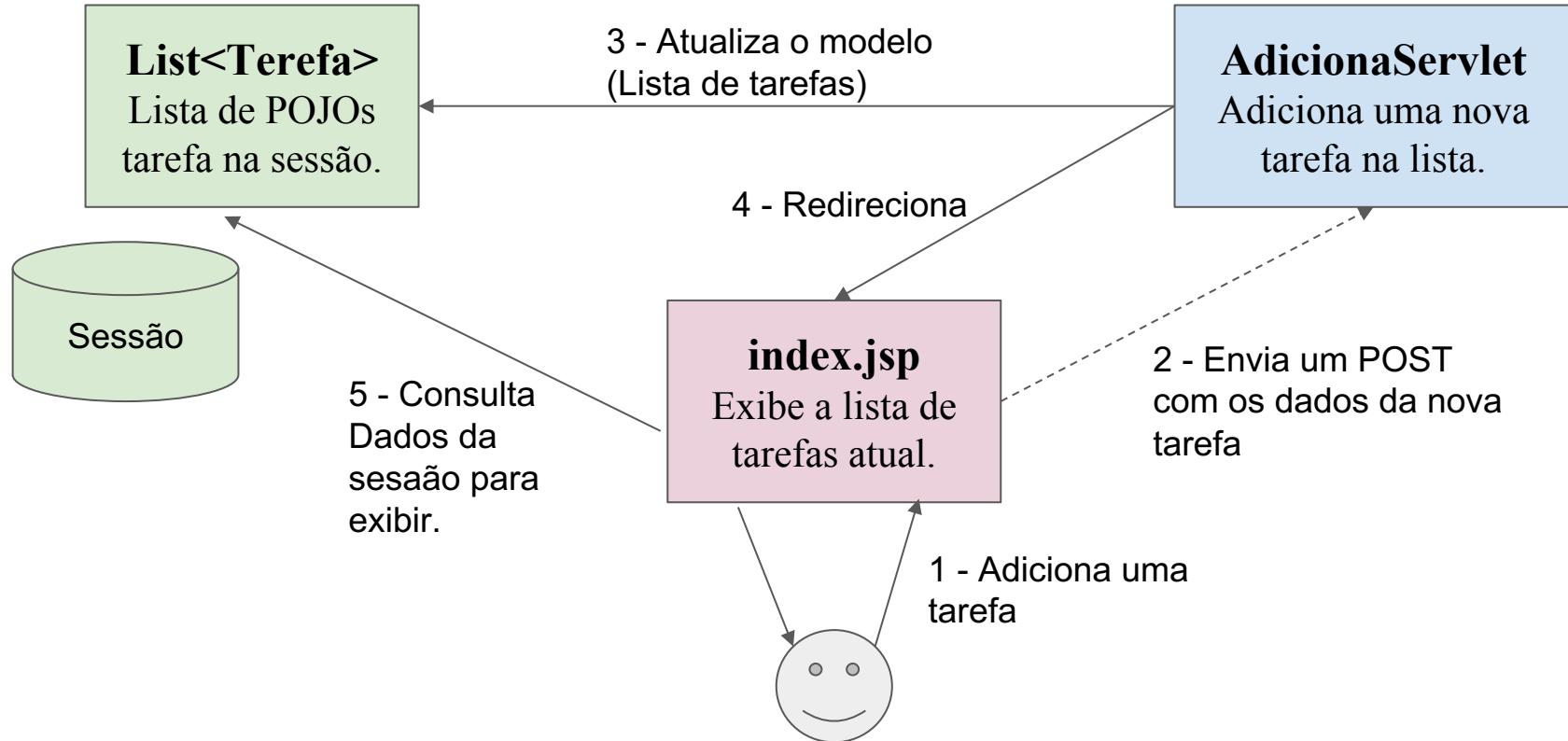
Recupera a sessão.

Atualiza o modelo  
(Cria nova tarefa e adiciona nas tarefas existentes.)

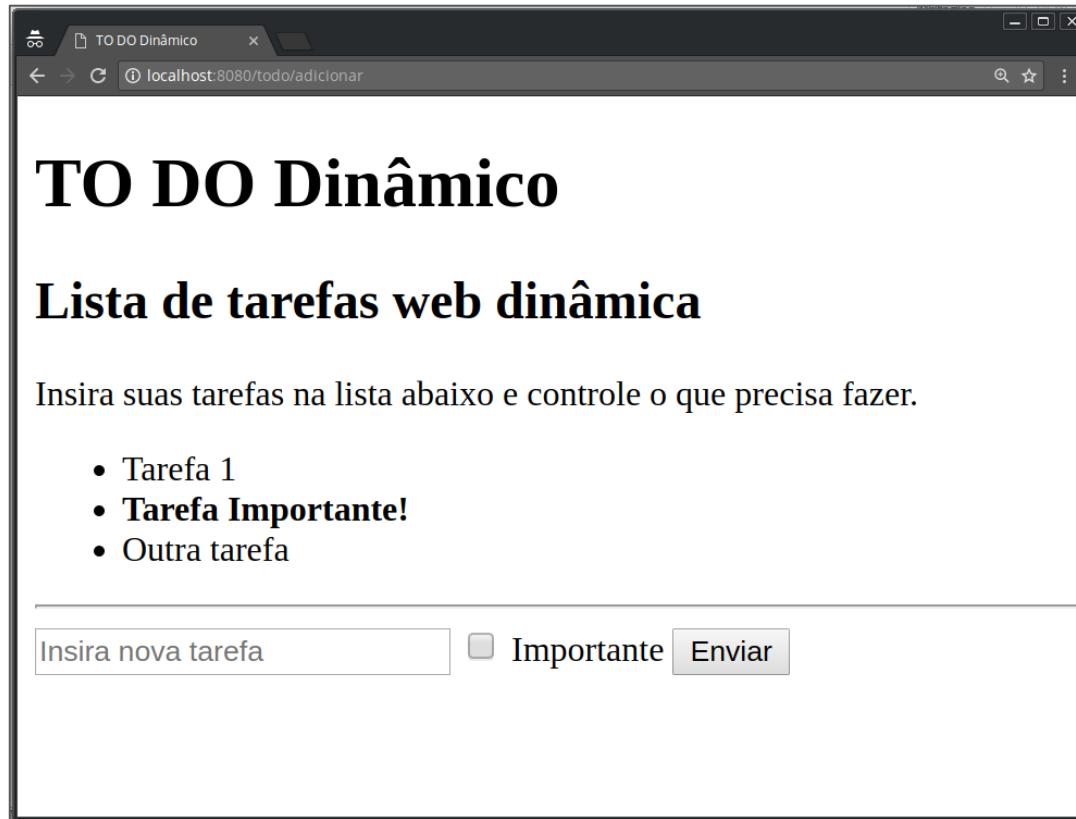
Atualiza a sessão.

Com o modelo atualizado, passa o controle para a View.

# JSP | Servlets + JSP | MVC



# | JSP | Servlets + JSP | MVC | Modelo



# | JSP | Scriptlets

□ Utilizando scriptlet o MVC ainda ficou “espaguete”.

```
<%@page import="java.util.List"%>
<%@page import="todo.model.Tarefa"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head> ... </head>
    <body>
        <div class="centro">
            <h1>TO DO Dinâmico</h1>
            <h2>Lista de tarefas web dinâmica </h2>
            <p>Insira suas tarefas na lista abaixo e controle o que precisa fazer.</p>
            <ul>
                <% List<Tarefa> tarefas = (List<Tarefa>) session.getAttribute("tarefas"); %>
                <% if(tarefas != null) {%
                    <% for (Tarefa tarefa : tarefas) { %
                        <% if(tarefa.getImportante()) {%
                            <li><strong><%= tarefa.getDescricao() %></strong></li>
                        <% } else {%
                            <li><%= tarefa.getDescricao() %></li>
                        <% }%
                    <% }%
                <% }%
            </ul>
            <hr />
            <form action="/todo/adicionar" method="post">
                <input type="text" name="descricao" placeholder="Insira nova tarefa" />
                <input type="checkbox" name="importante"/> Importante <input type="submit" name="enviar" value="Enviar" />
            </form>
        </div>
    </body>
</html>
```

Esse código ainda  
não ficou bom!

# | JSP | Scriptlets

- ❑ Porque separar código de negócio da visão?
  - ❑ Facilitar a manutenção da visão.
  - ❑ Permitir que uma equipe focada em frontend dê manutenção sem saber Java.
  - ❑ Permitir uma alteração visual com menor impacto.



# | JSP | JSTL

- ❑ Para evitar Java no JSP a Sun sugeriu o uso da **JavaServer Pages Standard Tag Library**, a **JSTL**.
  - ❑ A JSTL encapsula tags simples funcionalidades como controle de laços (fors), controle de fluxo do tipo if else, formatação, etc.
  - ❑ A JSTL foi a forma encontrada de padronizar o trabalho de milhares de programadores de páginas JSP.
- ❑ O primeiro passo para utilizar JSTL é importá-lo na página:

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

# | JSP | JSTL | Expression Language

- ❑ Juntamente com o JSTL podemos utilizar a **Expression Language (EL)**, que simplifica o acesso às informações.
  - ❑ Possuem a forma  `${ valor }` .
  - ❑ Podem imprimir informações na tela sem necessidade de `<%= ... %>`:
    - ❑  `${ valor }`
  - ❑ Podem acessar getters e setters diretamente.
    - ❑  `${ objeto.propriedade }`  e  `${ objeto.propriedade.prop }`
    - ❑  `${ objeto.propriedade = valor }`
  - ❑ Podem acessar valores de listas...
    - ❑  `${ lista[3] }`
  - ❑ ... e mapas:
    - ❑  `${ mapa["chave"] }`

# JSP | JSTL | Expression Language

```
<% List<Tarefa> tarefas = (List<Tarefa>)
session.getAttribute("tarefas"); %>
<% if(tarefas != null) {%
    <% for (Tarefa tarefa : tarefas) { %
        <% if(tarefa.getImportante()) {%
            <li>
                <strong>
                    <%= tarefa.getDescricao() %>
                </strong>
            </li>
        <% } else {%
            <li>
                <%= tarefa.getDescricao() %>
            </li>
        <% }%>
    <% }%>
<% }%>
```

```
<% List<Tarefa> tarefas = (List<Tarefa>)
session.getAttribute("tarefas"); %>
<% if(tarefas != null) {%
    <% for (Tarefa tarefa : tarefas) { %
        <% if(tarefa.getImportante()) {%
            <li>
                <strong>
                    ${tarefa.descricao}
                </strong>
            </li>
        <% } else {%
            <li>
                ${tarefa.descricao}
            </li>
        <% }%>
    <% }%>
<% }%>
```

# | JSP | JSTL | Condicionais

- ❑ Utilizando JSTL e EL podemos adicionar tags para exibição condicional.

```
<c:if test="${tarefa.importante}">  
....  
</c:if>
```

```
<c:choose>  
  <c:when test="${tarefa.importante}">  
    ...  
  </c:when>  
  <c:when test="${tarefa.descricao == null}">  
    ...  
  </c:when>  
  <c:otherwise>  
    ...  
  </c:otherwise>  
</c:choose>
```

# JSP | JSTL | Condicionais

```
<% List<Tarefa> tarefas = (List<Tarefa>)
session.getAttribute("tarefas"); %>
<% if(tarefas != null) {%
    <% for (Tarefa tarefa : tarefas) { %
        <% if(tarefa.getImportante()) {%
            <li>
                <strong>
                    <%= tarefa.getDescricao() %>
                </strong>
            </li>
        <% } else {%
            <li>
                <%= tarefa.getDescricao() %>
            </li>
        <% }%>
    <% }%>
<% }%>
```

```
<% List<Tarefa> tarefas = (List<Tarefa>)
session.getAttribute("tarefas"); %>
<% if(tarefas != null) {%
    <% for (Tarefa tarefa : tarefas) { %
        <li>
            <c:choose>
                <c:when test="${tarefa.importante}">
                    <strong>${tarefa.descricao}</strong>
                </c:when>
                <c:otherwise>
                    <li>${tarefa.descricao}
                </c:otherwise>
            </c:choose>
        </li>
    <% }%>
<% }%>
```

# | JSP | JSTL | Laços

- ❑ Utilizando JSTL e EL podemos adicionar tags para exibição de laços.

```
<table>
    <!-- percorre contatos montando as linhas da tabela -->
    <c:forEach var="contato" items="${contatos}">
        <tr>
            <td>${contato.nome}</td>
            <td>${contato.email}</td>
            <td>${contato.endereco}</td>
        </tr>
    </c:forEach>
</table>
```

# JSP | JSTL | Condicionais

```
<% List<Tarefa> tarefas = (List<Tarefa>)
session.getAttribute("tarefas"); %>
<% if(tarefas != null) {%
    <% for (Tarefa tarefa : tarefas) { %
        <% if(tarefa.getImportante()) {%
            <li>
                <strong>
                    <%= tarefa.getDescricao() %>
                </strong>
            </li>
        <% } else {%
            <li>
                <%= tarefa.getDescricao() %>
            </li>
        <% }%>
    <% }%>
<% }%>
```

```
<c:forEach var="tarefa"
           items="${sessionScope.tarefas}">
    <li>
        <c:choose>
            <c:when test="${tarefa.importante}">
                <strong>${tarefa.descricao}</strong>
            </c:when>
            <c:otherwise>
                ${tarefa.descricao}</li>
            </c:otherwise>
        </c:choose>
    </li>
</c:forEach>
```

# JSP | JSTL | Escopo

```
<c:forEach var="tarefa"
           items="${sessionScope.tarefas}">
    <li>
        <c:choose>
            <c:when test="${tarefa.importante}">
                <strong>${tarefa.descricao}</strong>
            </c:when>
            <c:otherwise>
                ${tarefa.descricao}</li>
            </c:otherwise>
        </c:choose>
    </li>
</c:forEach>
```

- ❑ Existem diferentes escopos acessíveis na EL.
  - ❑ **pageScope** escopo do JSP atual.
  - ❑ **requestScope** escopo da requisição atual.
  - ❑ **sessionScope** escopo da sessão atual.
  - ❑ **applicationScope** escopo da aplicação atual.

# JSP | JSTL | Escopo

```
<c:forEach var="tarefa"
           items="${tarefas}">
    <li>
        <c:choose>
            <c:when test="${tarefa.importante}">
                <strong>${tarefa.descricao}</strong>
            </c:when>
            <c:otherwise>
                ${tarefa.descricao}</li>
            </c:otherwise>
        </c:choose>
    </li>
</c:forEach>
```

- ❑ Se não informarmos o escopo, o atributo será buscado na ordem de precedência dos escopos.
- ❑ Pode haver conflito de nomes e bugs em função disso.

# | JSP | Servlets + JSP | Atividade

- ❑ Implemente uma página para avaliação de fotos a partir do modelo/template disponibilizado.
  - ❑ Por enquanto, as informações serão armazenadas na sessão.
- ❑ Para isso crie 2 páginas e quantos servlets forem necessários:
  - ❑ **Página 1:** Página com a postagem de fotos e listagem de avaliações.
  - ❑ **Página 2:** Cadastro de novas avaliações.
- ❑ Crie os servlets que achar necessários.
- ❑ Lembre-se de seguir o padrão MVC e usar JSTL na visão.
- ❑ Envie para [tassio@tassio.eti.br](mailto:tassio@tassio.eti.br) com o assunto Atividade 6 - PhotArt JSP até XX/XX

The image consists of two screenshots of a web application called PhotArt. The top screenshot shows a circular planter on a spiral staircase with several green plants. Below the image is a rating bar with four yellow stars and one empty star, followed by the text 'Avaliação 4.3 / 5'. The bottom screenshot shows the same image with an 'Add Review' form overlaid. The form includes fields for 'Title' (filled with 'Muito bom!'), 'User' (filled with '@johndoe'), 'Review' (empty), and 'Rating' (a row of five radio buttons). A large blue button at the bottom right says 'Publicar Avaliação'.

# JSP | Autenticação

- ❑ É um requisito comum em aplicações web autenticar os usuários para que eles possam fazer alterações nos dados da aplicação.
  - ❑ Cliente em um sistema de compras.
  - ❑ Responsável por uma ação em um sistema de controle.
  - ❑ Nível de acesso do usuário para as informações.
- ❑ Existem vários mecanismos de autenticação:
  - ❑ Usuário e Senha.
  - ❑ Autenticação em 2 passos.
  - ❑ Tokens
  - ❑ Biometria



# JSP | Autenticação vs Autorização

- ❑ É como que haja confusão entre os conceitos **autenticação e autorização**.
  - ❑ Autenticação:
    - ❑ Processo de verificação de uma identidade alegada, por meio de comparação das credenciais apresentadas pelo usuário com outras pré-definidas.
  - ❑ Autorização:
    - ❑ Processo que ocorre após a autenticação e que tem a função de diferenciar os privilégios atribuídos ao sujeito autenticado. c



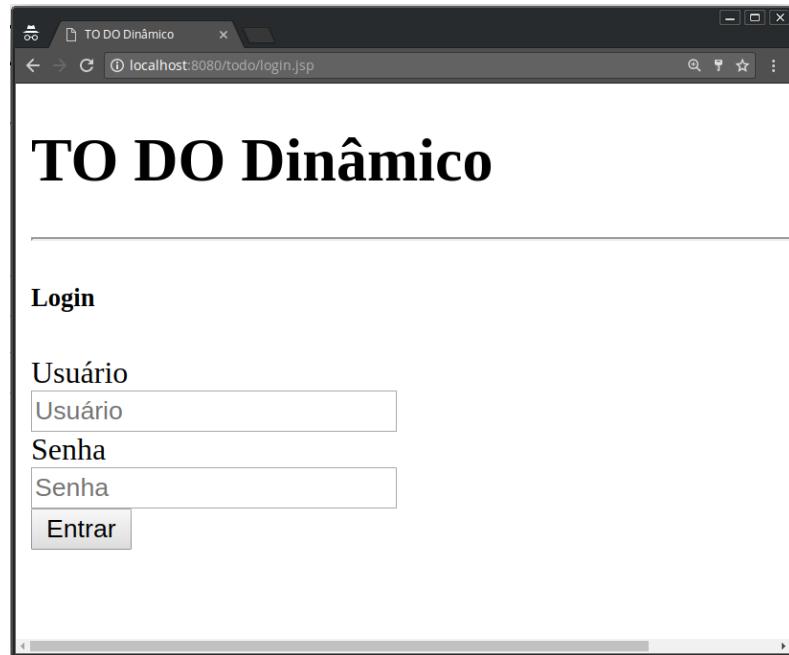
# | JSP | Utilizando Servlets e JSPs para Autenticação

- ❑ Vamos estudar como implementar mecanismos simples de autenticação utilizando Servlets e JSPs:
  - a. Adicionando uma página para verificação se o usuário está autenticado.
  - b. Criar um servlet que vai gravar na sessão as informações do usuário autenticado através desta página.
  - c. Redirecionar para a página de login quando não houver informação de autenticação na sessão em algum outro controller ou página.



# JSP | Autenticação

❑ Implementação de autenticação:



TO DO Dinâmico

Login

Usuário

Senha

Entrar



TO DO Dinâmico

## Lista de tarefas web dinâmica

Insira suas tarefas na lista abaixo e controle o que precisa fazer.

- Tarefa 1
- **Tarefa Importante!**
- Outra tarefa

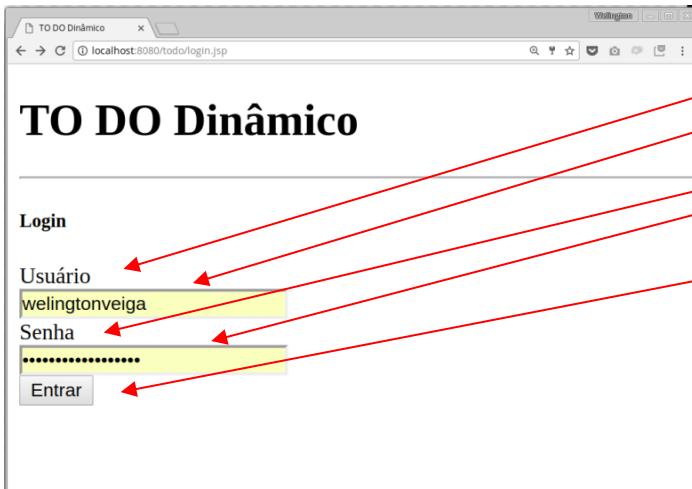
---

Insira nova tarefa

 Importante

# JSP | Autenticação | 1 - Página de Login

- Criamos um formulário de login, que deve ser informado no primeiro acesso do usuário.



```
login.jsp

<html>
  <head> ... </head>
  <body>
    <div class="centro">
      <h1>TO DO Dinâmico</h1>
      <hr />
      <h5>Login</h5>
      <form action="/todo/login" method="post">
        <label for="username">Usuário</label> <br />
        <input id="username" type="text"
               name="username" placeholder="Usuário" /><br />
        <label for="password">Senha</label> <br />
        <input id="password" type="password"
               name="password" placeholder="Senha"/><br />
        <input type="submit" name="entrar" value="Entrar" />
      </form>
    </div>
  </body>
</html>
```

# JSP | Autenticação | 2 - Definimos o modelo de usuário

Usuario.java

```
public class Usuario {  
  
    private String name;  
    private String password;  
  
    public Usuario(String name, String password) {  
        this.name = name;  
        this.password = password;  
    }  
  
    // Getters e setters  
  
}
```

```
@Override  
public int hashCode() {  
    return // ...;  
}  
  
}
```

```
@Override  
public boolean equals(Object obj) {  
    return // ...;  
}  
  
@Override  
public String toString() {  
    return "Usuario{" + "name=" + name + ", password=" + password + '}';  
}  
}
```

O modelo de usuário precisa ter informações para identificação e autenticação do mesmo.  
Muitas vezes precisa estar relacionado à informações de autorização.  
Um usuário pode ou não estar ligado a uma outra entidade, como pessoa, cliente, aluno, etc.

Métodos definidos no contrato de Object precisam estar definidos:  
equals, hashCode e toString devem ser implementados.

# JSP | Autenticação | 3 - Servlet de Login

LoginController.java

```
@WebServlet(urlPatterns = "/login")
public class LoginController extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        // 1 - Garantimos que a sessão existe.
        HttpSession session = req.getSession(true);

        // 2 - Verificamos se o usuário existe.
        String username = req.getParameter("username");
        String password = req.getParameter("password");

        // 3 - Verificamos se reconhecemos o usuário.
        // Poderia ser um banco de dados, serviço externo, etc...
        if (username.equals("admin") && password.equals("admin")) {
            // 4 - Salvamos o login
            Usuario usuario = new Usuario(username, password);
            session.setAttribute("usuario", usuario);
            RequestDispatcher rd = req.getRequestDispatcher("/index.jsp");
            rd.forward(req,resp);
        } else {
            req.setAttribute("erro", "Usuário ou senha desconhecido.");
            RequestDispatcher rd = req.getRequestDispatcher("/login.jsp");
            rd.forward(req,resp);
        }
    }
}
```

Garantimos que  
há sessão.

Obtemos da requisição  
os dados usuário e senha.

Se os dados recebidos  
são válidos, redirecionamos  
para a página inicial.

Caso contrário retornamos  
para a página de login  
com uma mensagem de erro.

# JSP | Autenticação | 4 - Restringir o acesso não autenticado

- ❑ Mas como restringir o acesso à páginas?
  - ❑ Verificando o parâmetro “usuario” na sessão.
  - ❑ Podemos adicionar essa verificação em todos os servlets menos no servlet que processa a autenticação.

```
@WebServlet(urlPatterns = "/adicionar")
public class AdicionarController extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        // 1 - Garantimos que a sessão existe.
        HttpSession session = req.getSession(true);
        if (session.getAttribute("usuario") == null) {
            req.setAttribute("erro", "Favor realize login para continuar.");
            RequestDispatcher rd = req.getRequestDispatcher("/login.jsp");
            rd.forward(req, resp);
        }

        // ... Regras do Servlet ...

        RequestDispatcher rd = req.getRequestDispatcher("/index.jsp");
        rd.forward(req, resp);
    }
}
```

# JSP | Autenticação | 4 - Restringir o acesso não autenticado

## ❑ E para restringir o acesso à página JSP?

- ❑ Podemos criar um servlet que decide se redireciona para a página ou não.

```
IndexController.java
@WebServlet(name = "IndexController", urlPatterns = {" /index"})
public class IndexController extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        if (session != null && session.getAttribute("usuario") != null) {
            RequestDispatcher rd = request.getRequestDispatcher("/index.jsp");
            rd.forward(request, response);
        } else {
            request.setAttribute("erro", "Usuário ou senha desconhecido.");
            RequestDispatcher rd = request.getRequestDispatcher("/login.jsp");
            rd.forward(request, response);
        }
    }
}
```

# JSP | Autenticação | 4 - Restringir o acesso não autenticado

- Para que o servlet seja carregado antes da página, podemos utilizar o web.xml.

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... >

<welcome-file-list>
    <welcome-file>index</welcome-file>
</welcome-file-list>
```

Quando a URL base da aplicação é informada, o glassfish procura pelas páginas iniciais padrão:  
(index.jsp, index.html ...)

Usando a tag **welcome-list-files** é possível definir seu próprio JSP/Servlet inicial.  
URL Pattern do servlet.  
Atenção, sem "/"

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
</web-app>
```

# JSP | Autenticação | 5 - Feedback

- ❑ Vamos imprimir a mensagem de erro na página de login!

Web.xml

```
<div class="centro">
    <h1>TO DO Dinâmico</h1>
    <hr />
    <h5>Login</h5>
    <c:if test="${not empty requestScope.erro}">
        <p style="color: red">${requestScope.erro}</p>
    </c:if>
    <p>Entre com suas credenciais</p>
    <form action="/todo/login" method="post">
        <label for="username">Usuário</label> <br />
        <input id="username" type="text" name="username" placeholder="Usuário" /><br />
        <label for="username">Senha</label> <br />
        <input id="password" type="password" name="password" placeholder="Senha"/><br />
        <input type="submit" name="entrar" value="Entrar" />
    </form>
</div>
```

Imprimimos um feedback de erro, caso haja.

# JSP | Autenticação

## ❑ Exemplo de autenticação:



TO DO Dinâmico

---

Login

Usuário

Senha

Entrar



TO DO Dinâmico

### Lista de tarefas web dinâmica

Insira suas tarefas na lista abaixo e controle o que precisa fazer.

- Tarefa 1
- **Tarefa Importante!**
- Outra tarefa

---

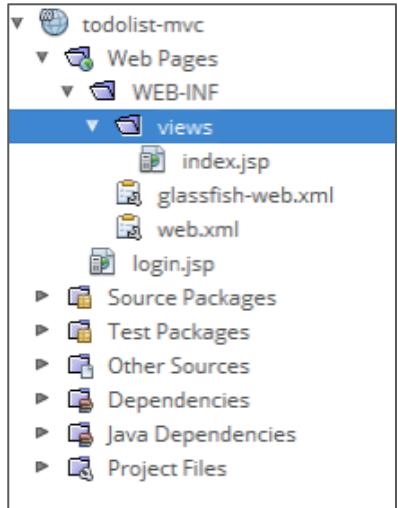
Insira nova tarefa

Importante

Enviar

# JSP | Autenticação | Atenção!

- ❑ Nesse exemplo os JSPs ainda podem ser acessados diretamente.
  - ❑ Uma dica para evitar isso é adicioná-los onde não podem ser acessados, como na pasta WEB-INF:



```
HttpSession session = req.getSession(true);
if (session.getAttribute("usuario") != null) {
    RequestDispatcher rd;
    rd = req.getRequestDispatcher("/WEB-INF/views/index.jsp");
    rd.forward(req, resp);
}
```

# | JSP | Autenticação | Atividade

- ❑ A partir da atividade anterior, utilizando [esse template de página de login](#):
  - a. Adicionar a página como **login.jsp** no projeto.
  - b. Crie um modelo de usuário.
  - c. Proteja os controladores existentes para aceitarem apenas autenticados.
  - d. Deixe a página de login como acesso padrão.
- ❑ Lembre-se de seguir o padrão MVC e usar JSTL na visão.
- ❑ Envie para [tassio@tassio.eti.br](mailto:tassio@tassio.eti.br) com o assunto Atividade 6 - PhotArt JSP até xx/xx.

The diagram illustrates the PhotArt application's user interface flow. It begins with a screenshot of the PhotArt homepage, which displays a circular garden scene with a central plant and several potted plants on a balcony. Below the image, there is a green button labeled "Adicionar Avaliação" (Add Review). To the right of the image, the text "Avaliação 4,3 / 5" is shown with five yellow stars. A large downward arrow is positioned to the right of this section. The next part of the diagram shows a screenshot of the same page, but the "Adicionar Avaliação" button has been clicked, turning red. A second large downward arrow is positioned below this. The final part of the diagram shows a screenshot of the review form. The "Avaliação" button is now a standard white button with black text. On the right side of the form, there is a section titled "Escreva sua avaliação" (Write your review) with input fields for "Título" (Title), "Usuário" (User), "Avaliação" (Rating), and "Nota final" (Final Grade). A "Publicar Avaliação" (Post Review) button is located at the bottom right of the form. A third large downward arrow is positioned to the right of the review form screenshot.

# HTTP | Referências adicionais

1. <https://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html>
2. <https://www.caelum.com.br/apostila-java-web/servlets/>
3. <http://blog.caelum.com.br/java-ee6-comecando-com-as-servlets-3-0/>
4. <https://www.caelum.com.br/apostila-java-web/appendice-topicos-da-servlet-api/>
5. [https://www.tutorialspoint.com/servlets/servlets-form-data.htm](https://www.tutorialspoint.com/servlets/servlets_form_data.htm)
6. [https://www.tutorialspoint.com/jsp/jsp\\_syntax.htm](https://www.tutorialspoint.com/jsp/jsp_syntax.htm)
7. [https://www.owasp.org/index.php/Authentication\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Authentication_Cheat_Sheet)
8. <https://martinfowler.com/articles/web-security-basics.html>



**CES/JF**  
*Centro de Ensino Superior  
de Juiz de Fora*

# Padrões de Projeto (Design Patterns)

Prof. MSc. Tassio Sirqueira

[tassio@tassio.eti.br](mailto:tassio@tassio.eti.br)

# O que é padrão de projeto?

- Padrão de projeto é a tradução de Design Pattern, e tem como objetivo resolver problemas que ocorrem com frequência dentro da orientação a objetos.
- Nenhum padrão de projeto é um esqueleto pronto e definitivo de solução para um problema do dia a dia, ele é um modelo.
- Os padrões podem ser estruturados grosseiramente em tres categorias diferentes.
  - Criacional;
  - Estrutural;
  - Comportamental;

# O que é GoF?

- GoF ou Gang of Four (Gangue dos quatro).
- A GoF é formada por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, que juntos escreveram um livro chamado Design Patterns: Elements of Reusable Object-Oriented Software em 1994. O livro ganhou muita popularidade e inspirou muitos outros depois.
- Eles estabeleceram 23\* padrões de projeto separados em 3 categorias, padrões de criação, estruturais e comportamentais.
- \* Padrões definidos originalmente pela GoF.

# Padrões de Criação

- Focados na criação de objetos
  - ✓ Abstract Factory\*
  - ✓ Builder\*
  - ✓ Factory Method\*
  - ✓ Multiton (Considerado um anti-pattern!)
  - ✓ Pool
  - ✓ Prototype\*
  - ✓ SimpleFactory
  - ✓ Singleton\* (Considerado um anti-pattern!)
  - ✓ StaticFactory

# Padrões Estruturais

- Focados na associação entre objetos
  - ✓ Adapter\*
  - ✓ Bridge\*
  - ✓ Composite\*
  - ✓ DataMapper
  - ✓ Decorator\*
  - ✓ DependencyInjection
  - ✓ Facade\*
  - ✓ FluentInterface
  - ✓ Flyweight\*
  - ✓ Proxy\*
  - ✓ Registry

# Padrões Comportamentais

- Focados nas interações entre objetos
  - ✓ Chain Of Responsibilities\*
  - ✓ Command\*
  - ✓ Interpreter\*
  - ✓ Iterator\*
  - ✓ Mediator\*
  - ✓ Memento\*
  - ✓ NullObject
  - ✓ Observer\*
  - ✓ Specification
  - ✓ State\*
  - ✓ Strategy\*
  - ✓ TemplateMethod\*
  - ✓ Visitor\*

# Outros

- A engenharia de software está sempre em busca de novas soluções para deixar os software melhores e com isso a descoberta de novos padrões ainda estão surgindo, como, por exemplo:
  - ✓ Delegation
  - ✓ ServiceLocator
  - ✓ Repository
  - ✓ EAV

# Exemplos de Implementação

- **Design Patterns em PHP:**
  - <https://github.com/domnikl/DesignPatternsPHP>
- **Design Patterns em Java:**
  - <https://github.com/MarcosX/Padr-es-de-Projeto>

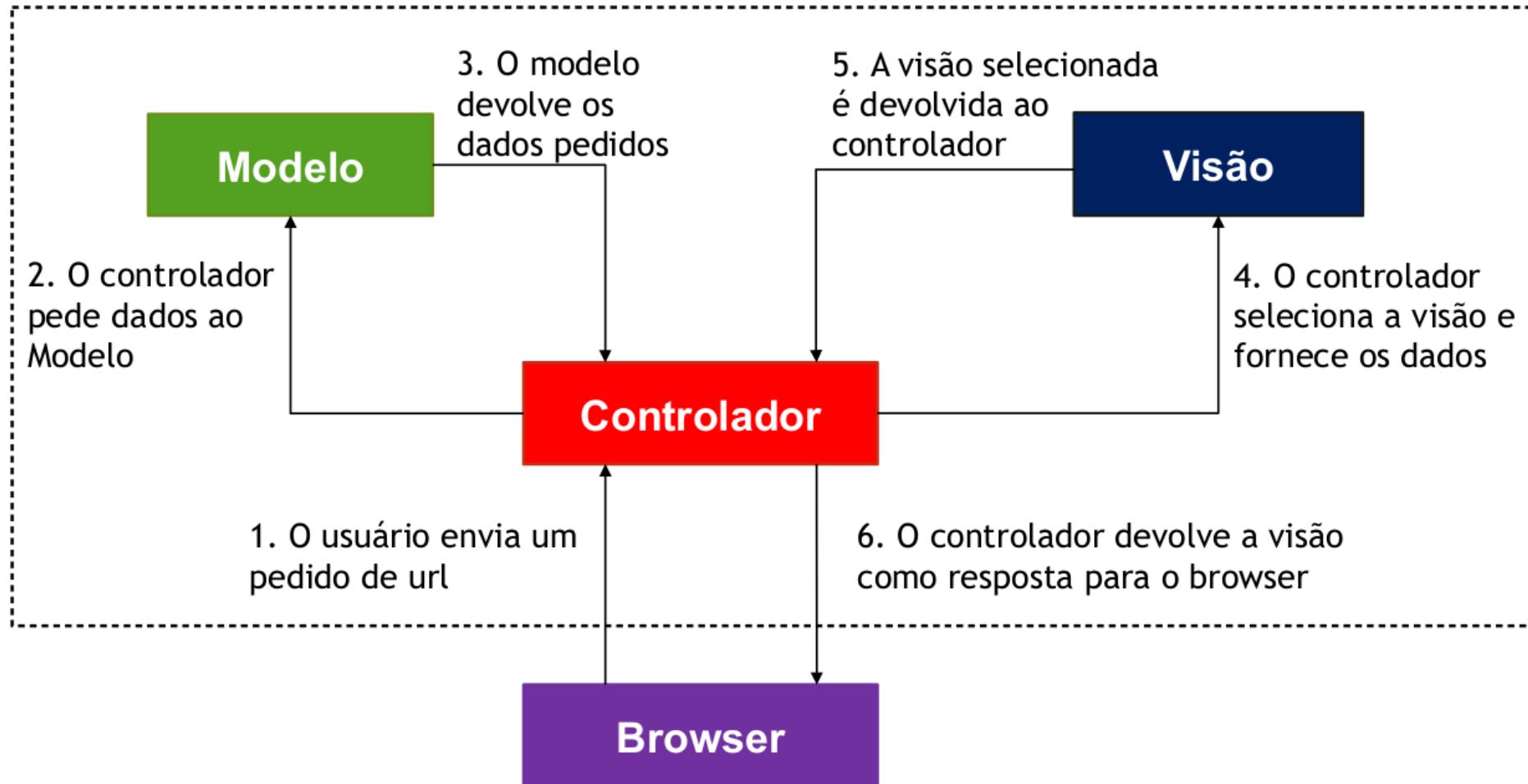
# Padrão Arquitetural MVC

- **Model-View-Controller (MVC)**
- É um padrão de arquitetura de aplicações que divide a aplicação em três camadas: a visão (view), o modelo (model), e o controlador (controller).
- O padrão MVC foi desenvolvido em 1979 por Trygve Reenskaug, com a finalidade de ser utilizado como arquitetura para aplicativos desktop. Entretanto, o padrão se popularizou para uso em sistemas web, a partir da adesão de milhares de Frameworks de mercado.

# O que é Arquitetura de Aplicação?

- A arquitetura de uma aplicação é a definição de suas estruturas.
- Dividir a aplicação em camadas: uma de interface do usuário denominada *View*, uma para manipulação lógica de dados, chamada *Model*, e uma terceira camada de fluxo da aplicação, chamada *Control*)
- Cria a possibilidade de exibir uma mesma lógica de negócios através de várias interfaces.
- Isolar a camada de negócios (*Model*) das demais camadas do sistema, de forma a facilitar a sustentabilidade do código.
- A implementação do controlador deve permitir que esta camada receba os eventos da interface e os converta em ações no modelo.

# Modelo MVC



# Padrão DAO (Data Access Object)

- O padrão DAO é um padrão de projeto que abstrai e encapsula os mecanismos de acesso a dados, escondendo os detalhes da execução da origem dos dados.
- O padrão de projeto DAO surgiu com a necessidade de separarmos a lógica de negócios da lógica de persistência de dados.
- Este padrão permite que possamos mudar a forma de persistência sem que isso influencie em nada na lógica de negócio, além de tornar nossas classes mais legíveis.

# Padrão DAO (Data Access Object)

- Este padrão permite criar as classes de dados independentemente da fonte de dados ser um BD relacional, um arquivo texto, um arquivo XML, JSON e etc.
- Para isso, ele encapsula os mecanismos de acesso a dados e cria uma interface de cliente genérica, para fazer o acesso aos dados, permitindo que os mecanismos de acesso sejam alterados, independentemente do código que utiliza os dados.

# Laboratório de Programação de Web Sites Dinâmicos

JavaEE + JPA

Tassio Sirqueira – 2019/02

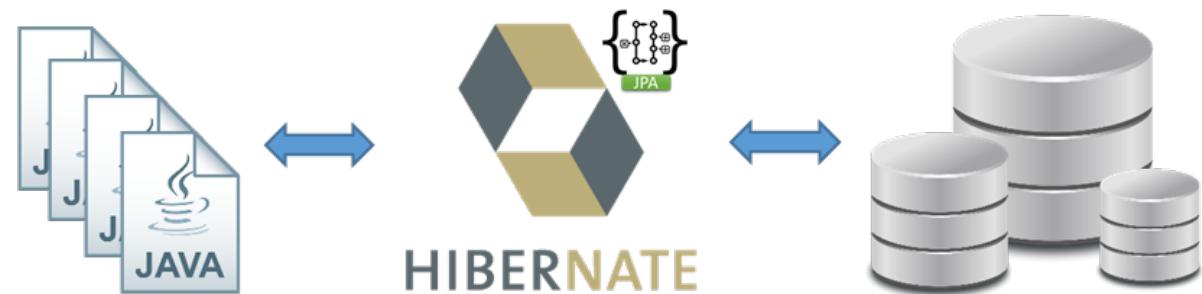
# | JavaEE + JPA | Introdução

- ❑ A integração com bancos de dados é um requisito comum para a maioria das aplicações.
- ❑ Com a popularização do Java no meio corporativo, observou-se o grande esforço dedicado à criação de SQL e manipulação JDBC.
- ❑ Além da produtividade foram observados outros problemas:
  - ❑ Diferença entre a sintaxe SQL de SGBDs diferentes.
  - ❑ Desafio no mapeamento objeto-relacional



# I | JavaEE + JPA | Introdução

- ❑ Diferentes alternativas para solucionar esse problema surgiram.
- ❑ Entre elas o **Hibernate** se popularizou, se tornando, desde então, líder de mercado.
- ❑ O hibernate é um framework ORM (Mapeamento Objeto Relacional), e busca resolver o que conhecemos como **impedância objeto** (representação seguindo princípios OO) **relacional** (representação normalizada).



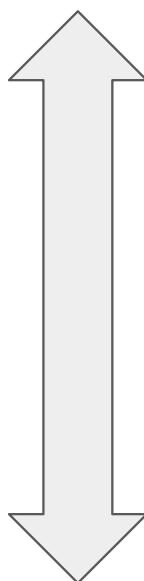
# I | JavaEE + JPA | Introdução

- ❑ Por sua popularidade e forte presença em diversos projetos Java, uma proposta de especificação de um framework inspirado no Hibernate foi proposto.
- ❑ O JPA, **Java Persistence API**, define uma série de classes, interfaces e padrões para a criação de frameworks ORM.
- ❑ O Hibernate é um dos frameworks que implementam o padrão JPA, assim como o **EclipseLink (referência)** e o OpenJPA.



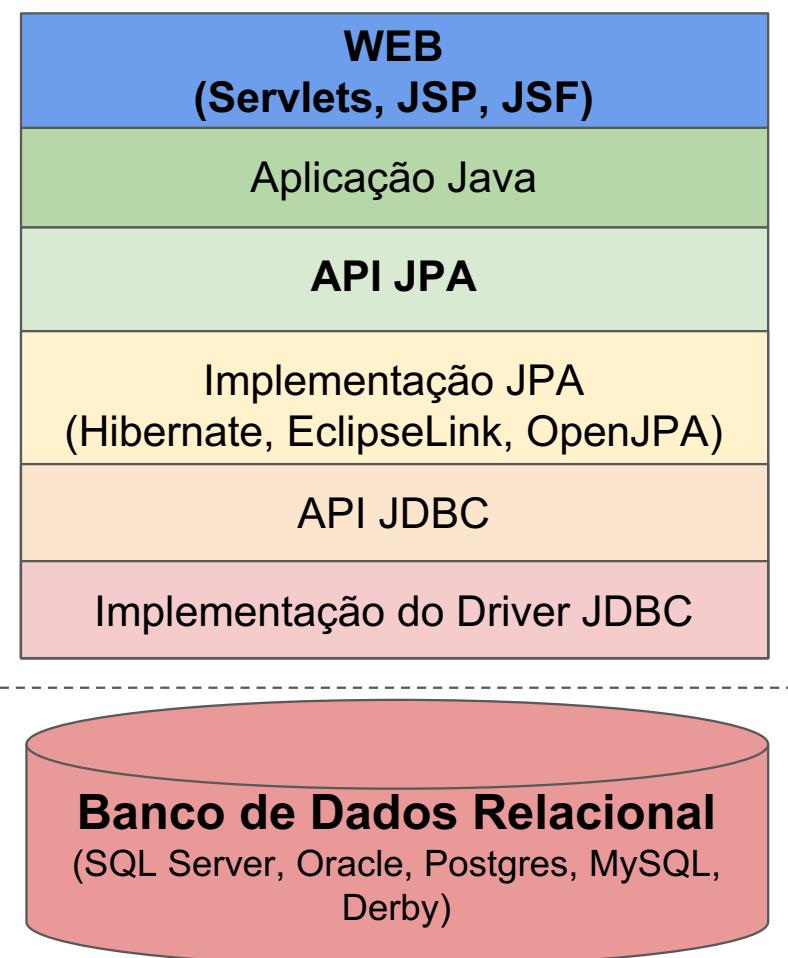
# | JavaEE + JPA | Arquitetura

Modelo Orientado à Objetos



Modelo Relacional

Servidor de Aplicação  
(Glassfish, Wildfly...)



# I JavaEE + JPA | Revisão de JPA

- ❑ Para utilizar o JPA, o primeiro passo é definir uma unidade de persistência (persistent unit).
- ❑ Deve ser definida no arquivo persistence.xml
  - ❑ Que deve ser estar na pasta META-INF, na raiz do classpath.
    - ❑ src/META-INF/persistence.xml
    - ❑ src/main/resources/META-INF/persistence.xml (**maven**)
- ❑ Uma unidade de persistência define:
  - ❑ As informações de conexão JDBC com o banco (URL, Driver...)
  - ❑ As informações necessárias para a implementação (Hibernate, EclipseLink...)

# JavaEE + JPA | Revisão de JPA | Exemplo de persistence.xml

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.1"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="estacionamento_unit" transaction-type="RESOURCE_LOCAL">
        <description> Hibernate JPA Estacionamento</description>
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:derby:estacionamento;create=true"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Definição da unidade de persistência. Uma aplicação java pode ter várias unidades de persistência.

Definição da implementação do JPA que deve ser utilizada nessa unidade de persistência. Nesse caso, a classe que define que utilizaremos o Hibernate é:  
**org.hibernate.jpa.HibernatePersistenceProvider**

# | JavaEE + JPA | Revisão de JPA | Exemplo de persistence.xml

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.1"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="estacionamento_unit" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="rop-and-create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Utilizando JPA a partir de um Servidor de Aplicações permite que as conexões e configurações de persistência sejam controladas por ele.

Assim como a escolha da implementação.

# I JavaEE + JPA | Revisão de JPA | Unidade de Persistência

- ❑ A partir da **Unidade de Persistência (*Persistence Unit*)**, acessamos outras classes essenciais para a JPA.
- ❑ A definição de unidades de persistência desacopla detalhes específicos da implementação das definições da especificação JPA.
- ❑ Se apenas o padrão da API for utilizado, é possível substituir o provider sem impactar outras partes da aplicação.
  - ❑ Na prática é muito difícil de fazer...



# I JavaEE + JPA | Revisão de JPA | Entidade

- ❑ Uma **entidade (*Entity*)** é um objeto que pode ser persistido no banco de dados.
- ❑ Essas classes possuem anotações especiais indicando **como devem ser persistidas** em um banco de dados relacional.
  - ❑ A partir dessas informações o JPA sabe como fazer o mapeamento objeto relacional entre as tabelas e colunas do banco de dados e os objetos e atributos em Java.
- ❑ Entidades estão relacionadas à uma linha de uma tabela no banco de dados.
  - ❑ Devem possuir identificadores únicos, e existem independentemente dos valores de seus atributos.

# JavaEE + JPA | Revisão de JPA | Exemplo de Entidade

- Definidos basicamente através da anotação `@Entity` e com uma coluna definida como `@Id`.

```
@Entity(name = "Product")
public class Product {

    @Id
    private Integer id;

    private String sku;

    private String name;

    private String description;

    // getters e setters omitidos.
}
```

# | JavaEE + JPA | Revisão de JPA | EntityManager

- ❑ Uma vez configurada, o acesso à API é feita pelo **EntityManager**.
  - ❑ Persistir/Atualizar objetos.
  - ❑ Recuperar uma entidade pelo tipo e pelo id.
  - ❑ Realizar uma consulta utilizando **JPQL (Java Persistence Query Language)** ou **SQL**.
  - ❑ Remover objetos.
- ❑ Operações feitas sobre objetos pelo **EntityManager** são automaticamente refletidas no banco de dados quando a transação é ‘commitada’.

# | JavaEE + JPA | Revisão de JPA | EntityManager

- ❑ Uma vez configurada, o acesso à API é feita pelo **EntityManager**.
  - ❑ Persistir/Atualizar objetos.
  - ❑ Recuperar uma entidade pelo tipo e pelo id.
  - ❑ Realizar uma consulta utilizando **JPQL (Java Persistence Query Language)** ou **SQL**.
  - ❑ Remover objetos.
- ❑ Operações feitas sobre objetos pelo **EntityManager** são automaticamente refletidas no banco de dados quando a transação é ‘commitada’.

# JavaEE + JPA | Revisão de JPA | Exemplo EntityManager

```
Person person = new Person();
entityManager.persist( person );

Phone phone = new Phone( "123-456-7890" );
phone.setPerson( person );
entityManager.persist( phone );

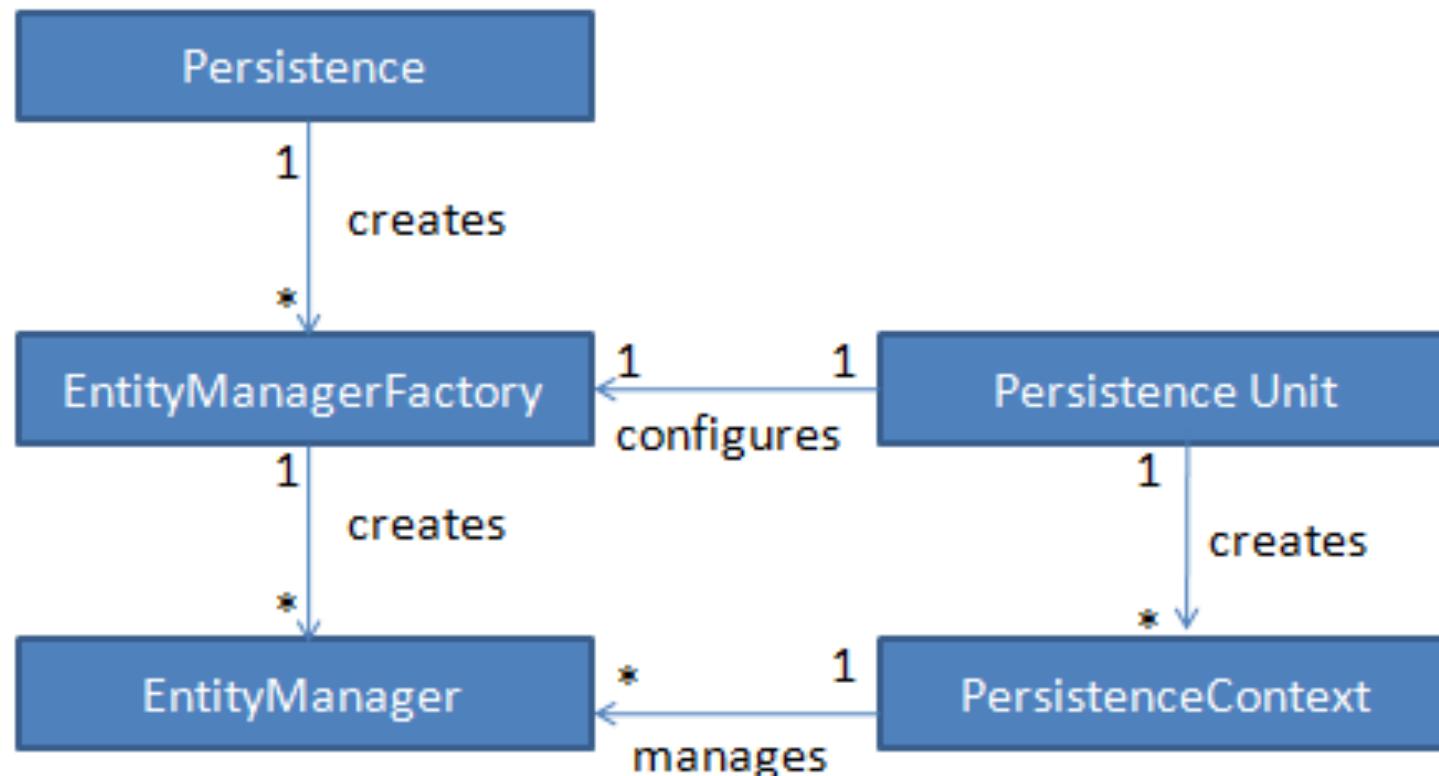
entityManager.flush();
phone.setPerson( null );
```

```
INSERT INTO Person ( id )
VALUES ( 1 )

INSERT INTO Phone ( number, person_id, id )
VALUES ( '123-456-7890', 1, 2 )

UPDATE Phone
SET      number = '123-456-7890',
        person_id = NULL
WHERE   id = 2
```

# JavaEE + JPA | Revisão de JPA



# I | JavaEE + JPA | Utilizando JPA no JavaEE

- ❑ Em um ambiente de aplicação gerenciado não é necessário instanciar e configurar o JPA manualmente.
  - ❑ O próprio servidor de aplicação configura o JPA.
- ❑ Podemos obter os objetos configurados do **contexto** da aplicação.
- ❑ Para isso é utilizado o padrão **Injeção de Dependência** por meio de **Inversão de Controle**.



## JavaEE + JPA | Utilizando JPA no JavaEE | Inversão de Controle

- ❑ Quando uma classe instancia diretamente suas dependências, com o operador **new**, introduz alto acoplamento.
  - ❑ Conhece e depende de uma implementação concreta.
  - ❑ Não está preso apenas à interface, e sim à implementação.
  - ❑ Precisa conhecer os detalhes da classe para instanciá-la.
  - ❑ Precisa gerenciar o ciclo de vida dessa classe.
- ❑ Quando uma classe recebe no construtor, métodos ou diretamente nos atributos suas dependências, há inversão de controle.
  - ❑ Baixo acoplamento, depende apenas da interface da classe.
  - ❑ Não conhece detalhes de implementação / instanciação.
  - ❑ Ciclo de vida independente.

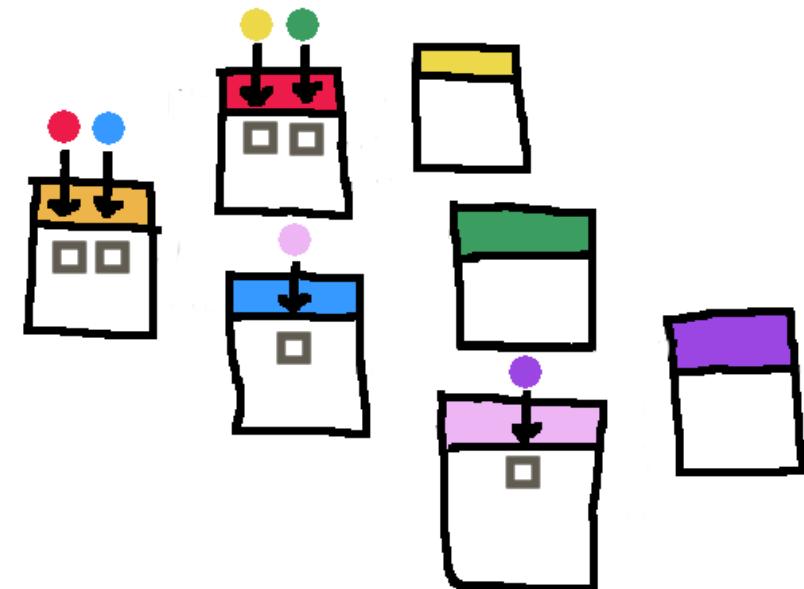
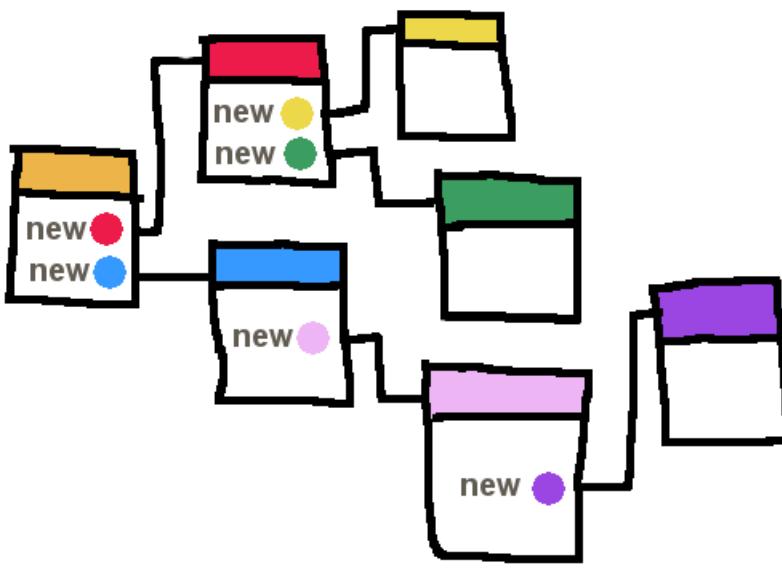
# JavaEE + JPA | Utilizando JPA no JavaEE | Inversão de Controle

```
class Relatorio {  
  
    private List<Dado> dados;  
  
    public Relatorio(List<Dado> dados){  
        this.dados = dados;  
    }  
  
    public void gerar() {  
  
        try (OutputStream out =  
            new BufferedOutputStream(  
                new FileOutputStream(OUTPUT_FILE), 1024)){  
            out.write(bytes);  
            for (Dado dado : dados){  
                out.write(dado.getBytes());  
            }  
            out.flush();  
        } catch (IOException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

```
class Relatorio {  
  
    private List<Dado> dados;  
  
    public Relatorio(List<Dado> dados){  
        this.dados = dados;  
    }  
  
    public void gerar(OutputStream out) {  
  
        try {  
            for (Dado dado : dados){  
                out.write(dado.getBytes());  
            }  
            out.flush();  
        } catch (IOException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

# JavaEE + JPA | Utilizando JPA no JEE | Injeção de Dependência

- ❑ Em um ambiente não gerenciado, temos controle direto dos objetos instanciados
- ❑ Em um ambiente gerenciado, as dependências são instanciadas e seu ciclo de vida controlado por um contexto.



# JavaEE + JPA | Utilizando JPA no JavaEE | Inversão de Controle

```
class Cofre {  
  
    private ControleAcess acesso;  
  
    public Relatorio(){  
        SQLControleAcesso sql =  
            new SQLControleAcesso();  
        sql.conecta();  
        this.acesso = sql;  
    }  
  
    public boolean podeAcessar(Credencial cred) {  
        return acesso.autoriza(cred);  
    }  
}
```

```
class Cofre {  
  
    @Inject  
    private ControleAcess acesso;  
  
    public boolean podeAcessar(Credencial cred) {  
        return acesso.autoriza(cred);  
    }  
}
```

# I JavaEE + JPA | Utilizando JPA no JavaEE

- ❑ Entre as tecnologias JEE, temos uma especificação para injeção de dependência, o **CDI**.
  - ❑ Não vamos estudar neste curso.
- ❑ Vamos utilizar conceitos de injeção em diversas especificações dentro da plataforma.



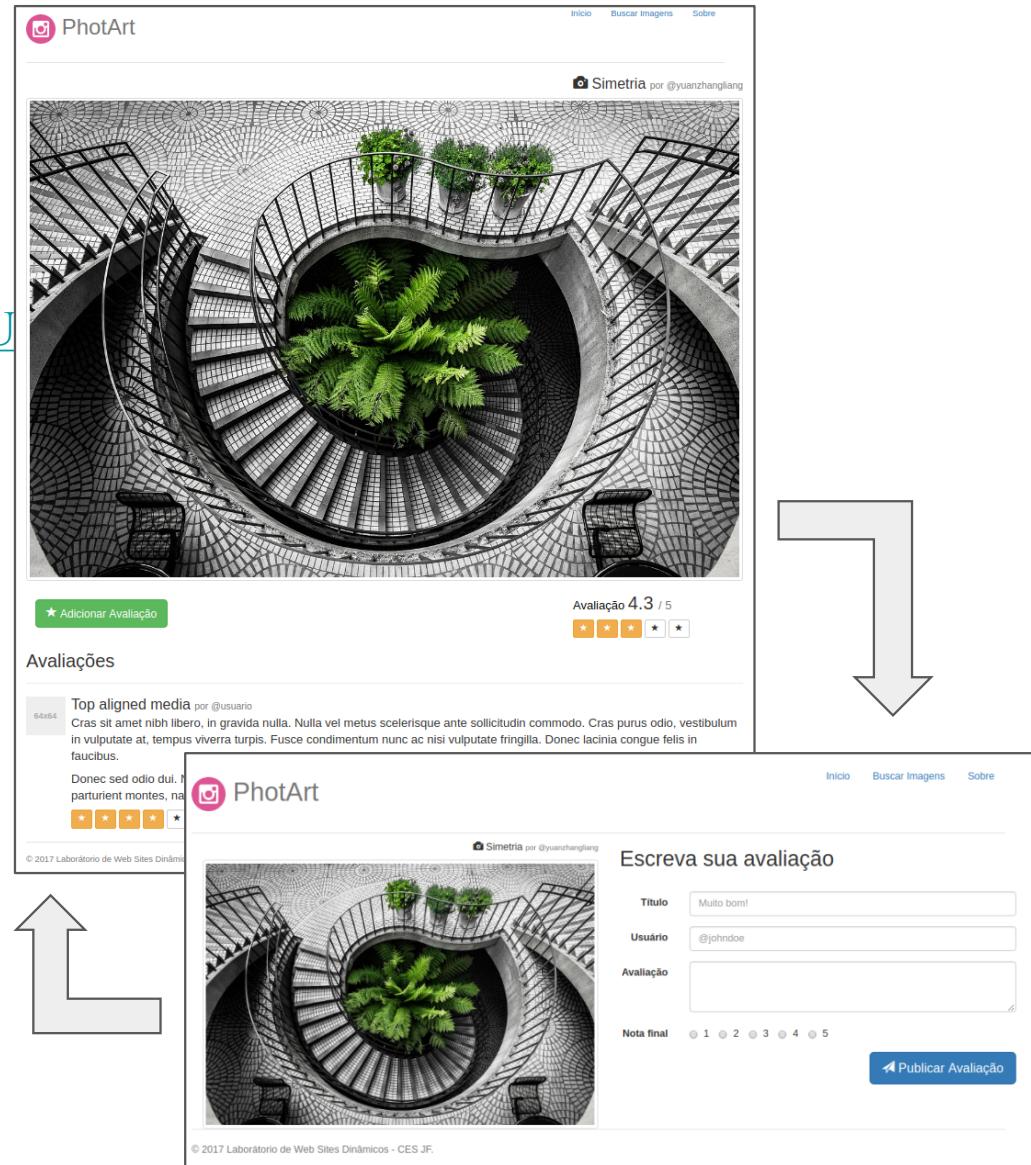
- ❑ Vamos alterar o projeto **todo list** para salvar as tarefas no banco de dados.
  - ❑ <https://drive.google.com/open?id=0B0c1aylmwuLUUF9qZ2lUUHFxaXM>
- ❑ Versão Alterada:
  - ❑ <https://drive.google.com/open?id=0B0c1aylmwuLUcW16TjFfZkxaeW8>



**NetBeans**

# | JavaEE + JPA | ATIVIDADE

- ❑ Utilizando o script  
<https://drive.google.com/file/d/0B0c1aylmwuLUU0NjM4V2FkZz0x/view?usp=sharing>
- ❑ Crie entidades, DAO e salve as avaliações no Banco de Dados.
- ❑ Envie para tassio@tassio.eti.br com o assunto Atividade 8 - PhotArt JPA até .



The image consists of two screenshots of the PhotArt website. The top screenshot shows a circular spiral staircase with green ferns in a planter, rated 4.3/5. The bottom screenshot shows the same image with an 'Add Review' form overlaid.

**Screenshot 1: PhotArt - Circular Spiral Staircase**

PhotArt

Simetria por @yuanzhangliang

Avaliação 4,3 / 5

★ Adicionar Avaliação

Avaliações

Top aligned media por @usuario

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin commodo. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

Donec sed odio dui. Nam portaurient montes, na

**Screenshot 2: PhotArt - Add Review Form**

PhotArt

Simetria por @yuanzhangliang

Escreva sua avaliação

Título: Muito bom!

Usuario: @johndoe

Avaliação:

Nota final: 1 2 3 4 5

Publicar Avaliação



# Data e Hora no Java

Prof. Tassio Sirqueira  
[tassio@tassio.eti.br](mailto:tassio@tassio.eti.br)

# Data e hora para máquinas

- A classe *Instant* é utilizada para o “agora”, com precisão de nanossegundos.
  - Instant agora = Instant.now();
  - System.out.println(agora); //2019-05-06T11:12:50.036Z (formato ISO-8601)
- Podemos usar um *Instant*, por exemplo, para medir o tempo de execução de um algoritmo.
  - Instant inicio = Instant.now();
  - Algoritmo();
  - Instant fim = Instant.now();
  - Duration duracao = Duration.between(inicio, fim);
  - long duracaoEmMilissegundos = duracao.toMillis();
- A classe *Duration* serve para medir uma quantidade de tempo em termos de nanossegundos. Essa quantidade de tempo pode ser alterada em diversas unidades, chamando métodos como *toNanos*, *toMillis*, *getSeconds*, etc.

# Datas para humanos

- A classe *LocalDate* que representa uma data, ou seja, um período de 24 horas com dia, mês e ano definidos.
  - `LocalDate hoje = LocalDate.now();`
  - `System.out.println(hoje); //2014-04-08 (formato ISO-8601)`
- Um *LocalDate* serve para representarmos datas em que não nos importamos com as horas ou minutos, mas o dia todo.
- Para calcularmos a duração entre dois *LocalDate*, devemos utilizar um *Period*, que já trata anos bissextos e outros detalhes.
  - `LocalDate homemNoEspaco = LocalDate.of(1961, Month.APRIL, 12);`
  - `LocalDate homemNaLua = LocalDate.of(1969, Month.MAY, 25);`
  - `Period periodo = Period.between(homemNoEspaco, homemNaLua);`
  - `System.out.printf("%s anos, %s meses e %s dias", periodo.getYears() , periodo.getMonths(), periodo.getDays()); //8 anos, 1 meses e 13 dias`

# Datas para humanos

- Já a classe *LocalTime* serve para representar apenas um horário, sem data específica. Podemos, por exemplo, usá-la para representar o horário de entrada no trabalho.
  - LocalTime horarioDeEntrada = LocalTime.of(9, 0);
  - System.out.println(horarioDeEntrada); //09:00
- A classe *LocalDateTime* serve para representar uma data e hora específicas.
  - LocalDateTime agora = LocalDateTime.now();
  - LocalDateTime aberturaDaCopa = LocalDateTime.of(2014, Month.JUNE, 12, 17, 0);
  - System.out.println(aberturaDaCopa); //2014-06-12T17:00 (formato ISO-8601)

# Datas com fuso horário

- Para representarmos uma data e hora em um fuso horário específico, devemos utilizar a classe `ZonedDateTime`.
  - `ZonedDateTime fusoHorarioDeSaoPaulo = ZonedDateTime.of("America/Sao_Paulo");`
  - `ZonedDateTime agoraEmSaoPaulo = ZonedDateTime.now(fusoHorarioDeSaoPaulo);`
  - `System.out.println(agoraEmSaoPaulo); //2014-04-08T10:02:57.838-03:00[America/Sao_Paulo]`

# Datas com fuso horário

- Com um ZonedDateTime, podemos representar, por exemplo, a data de um voo.
  - ZonedDateTime fusoHorarioDeSaoPaulo = ZonedDateTime.of("America/Sao\_Paulo");
  - ZonedDateTime fusoHorarioDeNovaYork = ZonedDateTime.of("America/New\_York");
  - LocalDateTime saidaDeSaoPauloSemFusoHorario = LocalDateTime.of(2014, Month.APRIL, 4, 22, 30);
  - LocalDateTime chegadaEmNovaYorkSemFusoHorario = LocalDateTime.of(2014, Month.APRIL, 5, 7, 10);
  - ZonedDateTime saidaDeSaoPauloComFusoHorario =  
ZonedDateTime.of(saidaDeSaoPauloSemFusoHorario, fusoHorarioDeSaoPaulo);
  - System.out.println(saidaDeSaoPauloComFusoHorario); //2014-04-04T22:30-  
03:00[America/Sao\_Paulo]
  - ZonedDateTime chegadaEmNovaYorkComFusoHorario =  
ZonedDateTime.of(chegadaEmNovaYorkSemFusoHorario, fusoHorarioDeNovaYork);
  - System.out.println(chegadaEmNovaYorkComFusoHorario); //2014-04-05T07:10-  
04:00[America/New\_York]
  - Duration duracaoDoVoo = Duration.between(saidaDeSaoPauloComFusoHorario,  
chegadaEmNovaYorkComFusoHorario);
  - System.out.println(duracaoDoVoo); //PT9H40M

# Datas com fuso horário

- Outro cuidado importante que devemos ter é em relação ao horário de verão. No fim do horário de verão, por exemplo, a mesma hora existe duas vezes!
  - ZonedDateTime fusoHorarioDeSaoPaulo = ZonedDateTime.of("America/Sao\_Paulo");
- - LocalDateTime fimDoHorarioDeVerao2013SemFusoHorario = LocalDateTime.of(2014, Month.FEBRUARY, 15, 23, 00);
- - ZonedDateTime fimDoHorarioVerao2013ComFusoHorario = fimDoHorarioDeVerao2013SemFusoHorario.atZone(fusoHorarioDeSaoPaulo);
  - System.out.println(fimDoHorarioVerao2013ComFusoHorario); //2014-02-15T23:00-02:00[America/Sao\_Paulo]
- - ZonedDateTime maisUmaHora = fimDoHorarioVerao2013ComFusoHorario.plusHours(1);
  - System.out.println(maisUmaHora); //2014-02-15T23:00-03:00[America/Sao\_Paulo]

# Formatando datas

- O *toString* padrão das classes da API utiliza o formato ISO-8601. Se quisermos definir o formato de apresentação da data, devemos utilizar o método *format*, passando um *DateTimeFormatter*.
  - LocalDate hoje = LocalDate.now();
  - DateTimeFormatter formatador = DateTimeFormatter.ofPattern("dd/MM/yyyy");
  - hoje.format(formatador); //08/04/2014

# Formatando datas

- O enum *FormatStyle* possui alguns formatos pré-definidos, que podem ser combinados com um *Locale*.
  - `LocalDateTime agora = LocalDateTime.now();`
  - `DateTimeFormatter formatador = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT).withLocale(new Locale("pt", "br"));`
  - `agora.format(formatador); //08/04/14 10:02`

# Referências

- Costa, R. **Data e Hora no Java 8 e no Java 9.** <https://medium.com/@racc.costa/data-e-hora-no-java-8-e-no-java-9-5f1e3fd8d560>. Acessado em 06/05/19
- Aquiles, A. e Ferreira, R. **Conheça a nova API de datas do Java 8.** <https://blog.caelum.com.br/conheca-a-nova-api-de-datas-do-java-8/>. Acessado em 06/05/19
- Normandes. **Introdução à nova API de Datas do Java 8.** <https://blog.algaworks.com/introducao-a-nova-api-de-datas-do-java-8/>. Acessado em 06/05/19