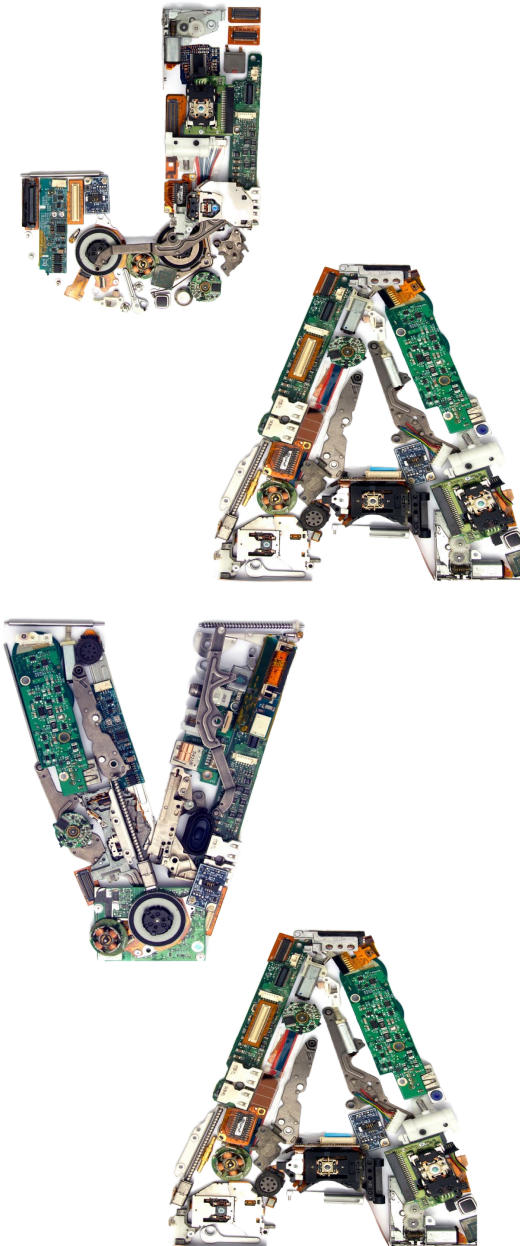


# Programação Orientada a Objetos com Java e WEB

JSP – Java Server Pages



# JSP – Java Server Page

**JavaServer Pages (JSP)** é uma tecnologia utilizada para criar páginas web dinâmicas baseadas em HTML, XML dentre outras tecnologias **no lado do servidor utilizando Java**. Ele permite a **inclusão de snippets de código Java diretamente no HTML, facilitando a criação de conteúdo dinâmico**.

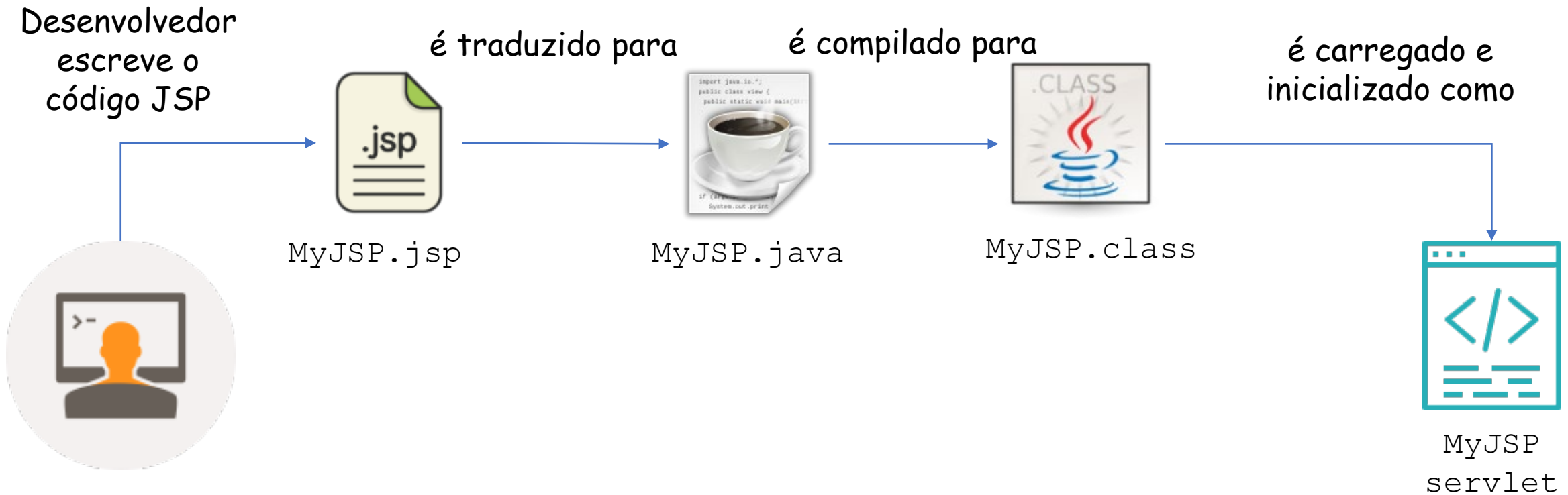
**JSP** tem semelhança com a linguagem **PHP**, mas usa a linguagem de programação Java. Como os **servlets**, para a execução do código **JSP** é necessário um servidor (**container**) web.

O **JSP** permite ao desenvolvedor de páginas web produzir aplicações que acessem banco de dados, manipulem arquivos no formato texto, capturem informações a partir de formulários e captem informações sobre o visitante e sobre o servidor.

**Uma página criada com a tecnologia JSP, após instalada em um servidor de aplicações compatível com a tecnologia Java EE, é transformada em um servlet.**

# JSP – Java Server Page

Um **JSP** torna-se um **servlet** rodando em uma aplicação. O código escrito no **JSP** é traduzido pelo **container** em uma classe **servlet**.



# JSP – Java Server Page

Toda página **JSP** possui a extensão **.jsp**. A página **JSP** é uma página **HTML** com codificação **Java**.

O código **Java** inserido na página **JSP** deve estar entre as tags **<%** e **%>**, que são chamadas de *scriptlets*.

Diferente dos **servlets**, as páginas **JSP** não precisam ser compiladas. Basta codificar a página e disponibilizar no container, que será responsável pela compilação em tempo de execução transformando o **JSP** em **bytecode**.


# JSP – Java Server Page

```
<html>
<head>
  <title>Minha Página JSP</title>
</head>
<body>
  <!-- Comentário em JSP -->
  <%
    String mensagem = "Bem-vindo a minha primeira página JSP";
  %>
  <% out.println(mensagem); %>
  <br>
  <% String desenvolvido = "Desenvolvido por Antonio Marcos Selmini"; %>
  <%= desenvolvido %>
</body>
</html>
```


fazendo comentários  
em JSP



*out* é um objeto  
implícito do JSP para  
fazer saída



existem várias possibilidades  
para fazer impressão do  
conteúdo de uma variável



# JSP – Java Server Page

## Contador.jsp

```
<%@ page import="br.fiap.entidade.*" %>
<html>
<title>Contador de Chamadas JSP</title>
<body>
<h1>Contador =
<%
    out.println(Contador.getContador());
%> </h1>
</body>
</html>
```

## Para importar múltiplos pacotes:

```
<%@ page import="br.fiap.entidade.*, java.util.*" %>
```

os nomes dos pacotes devem  
ser separado por vírgula e  
virem entre aspas

para utilizar uma classe externa é necessário fazer o import, como nos arquivos java. O import é realizado com a diretiva import e o nome do pacote entre aspas!!

## Contador.java

```
package br.fiap.entidade;
```

```
public class Contador {
    private static int contador;
```

```
    public static int getContador() {
        contador++;
        return contador;
    }
}
```

# JSP – Java Server Page

## Contador.jsp com out.println()

```
<%@ page import="br.fiap.entidade.*" %>
<html>
<title>Contador de Chamadas JSP</title>
<body>
<h1>Contador =
<%
    out.println(Contador.getContador());
%> </h1>
</body>
</html>
```

**nunca use um ponto e vírgula no final de uma expressão. A expressão é um argumento para out.println().**

## Contador.jsp com uso de expressão

```
<%@ page import="br.fiap.entidade.*" %>
<html>
<title>Contador de Chamadas JSP</title>
<body>
<h1>Contador =
    <%= Contador.getContador() %> </h1>
</body>
</html>
```

# JSP – Java Server Page

o container irá transformar esse código ...

... nesse código

```
<%= Contador.getContador() %>
```

```
<% out.println(Contador.getContador()); %>
```

```
<%= Contador.getContador(); %>
```

```
<% out.println(Contador.getContador()); %>
```

um ponto e vírgula  
nesse local ...

... significa isso aqui

nesse caso o código  
não irá compilar!!!




# JSP – Declarando variáveis

Como todo **JSP** é traduzido para um **servlet**, será que o conteúdo da classe **Contador** (slide 6) não poderia ser definido dentro do próprio **JSP**? É possível declarar variáveis dentro de um **JSP**? E métodos, podem ser definidos?

A resposta para as perguntas é sim!!!!

Para a declaração de uma variável no JSP é necessário que haja a exclamação depois da porcentagem.

Quando a exclamação é adicionada no scriptlet, a variável é declarada como variável de instância no servlet.



```
<html>
  <title>Contador de Chamadas JSP</title>
  <body>
    <%! int contador = 0; %>
    <h1>Contador = <%= ++contador %> </h1>
  </body>
</html>
```

# JSP – Declarando métodos

```
<html>
<title>Contador de Chamadas JSP</title>
<body>
    <%! int meuMetodo() {
        contador = contador * 3;
        return contador;
    }

    %>
    <%! int contador = 1; %>
    <h1>Contador = <%= meuMetodo() %> </h1>
</body>
</html>
```

não há nenhum problema  
em declarar a variável  
depois que ela é utilizada.

# JSP – gerando uma tabela HTML

```
<table>
    <%
        ContatoDAO dao = new ContatoDAO();
        List<Contato> contatos = dao.listar();

        for (Contato contato : contatos ) {
            <tr>
                <td><%=contato.getNome() %></td>
                <td><%=contato.getEmail() %></td>
                <td><%=contato.getEndereco() %></td>
                <td><%=contato.getNascimento() %></td>
            </tr>
        <% } %>
</table>
```

# Sessões – conceito

Sessões (**HTTP sessions**) são um mecanismo fundamental em aplicações web para armazenar informações relacionadas a um usuário entre múltiplas requisições HTTP. Em um ambiente web, o **protocolo HTTP é stateless** (sem estado), o que significa que cada requisição do cliente ao servidor é tratada de forma independente, sem informações sobre requisições anteriores. As sessões resolvem esse problema, permitindo que informações persistam entre requisições de um mesmo usuário.

## O que é uma Sessão?

Uma **sessão** é uma forma de manter dados relacionados ao usuário durante sua interação com a aplicação web. Ela armazena dados no lado do servidor e é associada a um cliente específico por meio de um identificador de sessão (geralmente armazenado em um cookie ou na URL).

# Sessões – estrutura básica

1. **Identificador de Sessão:** cada sessão tem um identificador único chamado **session ID**, que é enviado ao cliente e armazenado em um cookie ou anexado à URL.
2. **Armazenamento no Servidor:** Os dados de sessão são mantidos no servidor, e o **session ID** é usado para recuperar esses dados quando o cliente faz novas requisições.
3. **Duração:** sessões têm uma duração definida (por padrão, elas expiram após um período de inatividade), mas podem ser explicitamente encerradas pelo servidor ou cliente (por exemplo, quando o usuário faz logout).

# Sessões – criação e uso

Quando um usuário faz uma requisição ao servidor (por exemplo, ao fazer login), o servidor pode criar uma sessão para esse usuário usando **request.getSession()**. Isso cria um novo objeto de sessão. Um **session ID** único é gerado pelo servidor e enviado ao cliente, geralmente em um cookie chamado **JSESSIONID**.

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

Após a criação da sessão, você pode armazenar dados relacionados ao usuário dentro da sessão, como nome de usuário, status de login, carrinho de compras, etc. Isso é feito com métodos como **session.setAttribute()**. Esses atributos são armazenados no servidor.

```
session.setAttribute("username", "joao");
```

# Sessões – recuperação de dados

Em requisições subsequentes, o servidor reconhece o usuário pelo **session ID** (geralmente via cookie enviado pelo cliente) e pode recuperar os dados da sessão. Isso permite que informações persistam entre páginas, como mostrar o nome do usuário em várias páginas após o login.

```
HttpSession session = request.getSession(false); // Recupera a sessão existente
String username = (String) session.getAttribute("username");
```

# Sessões – encerramento da sessão

Uma sessão pode ser encerrada explicitamente pelo servidor chamando **session.invalidate()**. Isso remove todos os atributos e encerra a sessão para o usuário. Sessões também podem expirar automaticamente após um período de inatividade configurado.

```
session.invalidate(); // Encerrar a sessão explicitamente
```



# Sessões – ciclo de vida



O servidor cria a sessão quando necessário (ex.: ao fazer login).

Durante as interações subsequentes, o **session ID** identifica a sessão e seus dados são recuperados

A sessão expira após um período de inatividade configurado ou pode ser invalidada manualmente (ex.: ao fazer logout)

# Sessões – encerramento da sessão

A expiração padrão da sessão pode ser configurada no **web.xml** da aplicação ou dinamicamente no código. No **web.xml**:

```
<session-config>
    <session-timeout>30</session-timeout> <!-- Timeout em minutos -->
</session-config>
```

Dinamicamente em um servlet:

```
session.setMaxInactiveInterval(30 * 60); // Sessão expira em 30 minutos
```

# Filtros – conceito

O filtro (**Filter**) em Java EE é um componente que intercepta requisições e respostas em um ciclo de vida de uma aplicação web, antes ou depois que um recurso (como um **servlet**, **JSP** ou arquivo estático) seja acessado. Os filtros permitem realizar várias tarefas como validação, autenticação, logging, compressão de dados e muito mais.

## Conceito de Filtro

Um filtro em Java EE é usado para manipular a requisição e resposta HTTP que está entrando ou saindo de um recurso da aplicação. Eles são configurados para interceptar certas URLs ou tipos de requisição e podem ser aplicados de forma global ou seletiva.

# Filtros – características

**Interceptação de Requisições:** filtros interceptam as requisições HTTP antes que elas cheguem a um servlet, JSP ou outro recurso da aplicação.

**Manipulação de Requisições e Respostas:** filtros podem inspecionar e modificar tanto as requisições quanto as respostas. Por exemplo, eles podem adicionar cabeçalhos HTTP, redirecionar requisições ou comprimir o conteúdo de resposta.

**Cadeia de Filtros (Filter Chain):** filtros podem ser organizados em uma cadeia. Uma requisição passa por um filtro, que decide se passará a requisição adiante na cadeia ou não. Vários filtros podem atuar em sequência sobre a mesma requisição.

**Execução Antes e Depois:** filtros podem executar ações antes de um recurso ser acessado (requisição) e também depois que a resposta foi gerada (resposta).

**Reusabilidade:** filtros podem ser configurados para trabalhar com múltiplos recursos de uma aplicação (como vários servlets ou páginas JSP), oferecendo uma maneira modular de aplicar funcionalidade comum.

# Filtros – aplicações

Filtros têm muitos usos práticos em aplicações web. Aqui estão alguns dos mais comuns:

**Autenticação e Autorização:** verificar se o usuário está autenticado antes de permitir o acesso a uma página ou servlet. Um filtro pode impedir o acesso a páginas protegidas e redirecionar para uma página de login.

**Logging e Auditoria:** capturar e registrar informações sobre requisições e respostas, como tempo de resposta ou IP do cliente.

**Compressão de Conteúdo:** Compactar a resposta HTTP (por exemplo, usando GZIP) para reduzir o tamanho dos dados enviados ao cliente.

**Modificação de Cabeçalhos HTTP:** adicionar ou modificar cabeçalhos HTTP em requisições ou respostas, como configurar cabeçalhos de cache, política de segurança (CORS), ou configurar cookies.

# Referências

- ❑ DEITEL, H. M., DEITEL, P. J. **JAVA como programar**. 10ª edição. São Paulo: Prentice-Hall, 2010.

