

Ministério da Educação - MEC
Universidade Aberta do Brasil - UAB
Universidade Federal do Piauí - UFPI
Centro de Educação Aberta e a Distância - CEAD

DESENVOLVIMENTO PARA WEB

Vinicius Ponte Machado





Reitor
José Arimatéia Dantas Lopes

Vice-Reitora
Nadir do Nascimento Nogueira

Superintendente de Comunicação
Jacqueline Lima Dourado

Editor
Ricardo Alaggio Ribeiro

EDUFPI - Conselho Editorial
Ricardo Alaggio Ribeiro (presidente)
Antonio Fonseca dos Santos Neto
Francisca Maria Soares Mendes
José Machado Moita Neto
Solimar Oliveira Lima
Teresinha de Jesus Mesquita Queiroz
Viriato Campelo

Diretor do Centro de Educação Aberta e a Distância - CEAD
Gildásio Guedes Fernandes

Vice-Diretora do Centro de Educação Aberta e a Distância - CEAD
Lívia Fernanda Nery da Silva

Coordenadora do Curso de Licenciatura em Computação
Keylla Maria de Sá Urtiga Aita

Coordenador de Tutoria do Curso de Licenciatura em Computação
Aline Montenegro Leal Silva

EQUIPE TÉCNICA

Revisão de Originais
José Barbosa da Silva
Projeto Gráfico e Diagramação
Vilsselle Hallyne Bastos de Oliveira
Revisão Gráfica
Fabiana dos Santos Sousa



Dados internacionais de Catalogação na Publicação

M149d Machado, Vinicius Ponte.
Desenvolvimento para web / Vinicius Ponte
Machado. – Teresina : EDUFPI, 2019.
149 p.

ISBN

1. Desenvolvimento de sistemas web. 2. Internet.
3. Computação. I. Título.

CDD 004.69

De acordo com a Lei n. 9.9610, de 19 de fevereiro de 1998, nenhuma parte deste livro pode ser fotocopiada, gravada, reproduzida ou armazenada num sistema de recuperação de informações ou transmitida sob qualquer forma ou por qualquer meio eletrônico ou mecânico sem o prévio consentimento do detentor dos direitos autorais.

Editora da Universidade Federal do Piauí - EDUFPI
Campus Universitário Ministro Petrônio Portella
CEP: 64049-550 - Bairro Ininga - Teresina - PI - Brasil

APRESENTAÇÃO

Passados mais de 20 anos do advento da Internet, ainda se presencia um grande crescimento na audiência desse fenômeno de comunicação. Com isso, vemos uma mudança substancial de como as pessoas acessam a informação. Os programas, antes feitos para serem executados localmente na máquina do usuário dependiam da instalação e transporte de dados através de mídias físicas. Atualmente a grande maioria dos sistemas são do tipo baseados na web. Ou seja, as pessoas digitam um endereço eletrônico através de um navegador (browser) e acessam, via internet, os sistemas.

Para os programadores, o advento da internet ocasionou uma mudança na forma de como os sistemas são desenvolvidos. Algumas linguagens foram adaptadas, outras foram criadas para que os novos aplicativos pudessem tirar proveito da interconectividade, entre sistemas e dados, promovida pela Internet. Além disso, novas metodologias de desenvolvimento surgiram com base nesse paradigma. Nos dias atuais, temos linguagens já consolidadas e extremamente adaptáveis a diversos tipos de aplicações web.

O objetivo desta apostila é proporcionar um entendimento do desenvolvimento de sistemas web. O texto reúne conteúdo de diversos autores e contém as principais tecnologias utilizadas para o desenvolvimento de aplicações através da internet. Ela foi escrita de forma objetiva e cada capítulo é acompanhado de embasamento teórico, e ao final de cada unidade há exercícios. A bibliografia e a webliografia ao fim dos capítulos permite que o leitor se aprofunde na teoria apresentada em cada unidade uma vez que os temas aqui abordados não esgotam o assunto.

Na Unidade I apresentar-se-á os conceitos básicos relacionados ao tema e introduzir-se-ão as linguagens de formatação básicas para os sistemas baseados na web: HTML, XHTML e CSS. A Unidade II falará do modelo MVC e mostrará a Linguagem JAVA, uma das mais populares no que diz respeito a desenvolvimento para Internet. Por fim, a Unidade III traz as definições e conceitos da linguagem PHP, que atualmente é a mais utilizada para construção de páginas web dinâmicas.

Boa Leitura!!
Vinícius Machado

SUMÁRIO

Apresentação	3
CAPÍTULO 1	
1. Arquitetura de aplicações web	10
1.1 Arquitetura Cliente/Servidor x Sistema Baseado na Web	10
1.2 Servidores Web	11
1.3 Clientes Web	13
2. HTML	14
2.1 Marcadores	15
2.1.1 Cabeçalho	15
2.1.2 Parágrafos	16
2.1.3 Textos pré-formatados	16
2.1.4 Quebras de linha	17
2.1.5 Fonte	17
2.1.6 Caracteres especiais	20
2.2 Formulários	21
2.2.1 O Elemento <form>	22
2.2.2 Elementos <label>, <input>, e <textarea>	23
2.2.3 Elemento <button>	25
3. CSS	26
3.1 Método de aplicação do CSS no documento HTML	28
3.1.1 Método inline	28
3.1.2 Método interno	29
3.1.3 Método externo	29
3.2 Pseudoclasses e pseudoelementos	30
3.3 Propriedades	31
3.4 Unidades de medida	33
4 XHTML	34
4.1 Conceitos básicos	35
Exercícios	38
Webliografia	39

CAPÍTULO 2

5	MVC –Model View Controller.....	42
6	Linguagem JAVA	45
6.1	A Plataforma JAVA	47
6.2	Meu primeiro programa Java	48
6.3	Programação Básica em Java	49
6.3.1	Comentários.....	49
6.3.2	Tipos de Dados	49
6.3.3	Declaração de Variáveis.....	50
6.3.4	Conversões Entre Tipos	51
6.3.5	Constantes.....	51
6.3.6	Operadores.....	52
6.3.7	Incremento e Decremento	52
6.3.8	Operadores Relacionais e Booleanos	53
6.3.9	Strings	53
6.3.10	Vetores e Matrizes	54
6.3.11	Controle De Fluxo Do Programa	54
6.3.12	Outras instruções.....	57
6.4	Construção de Programas em Java.....	57
6.4.1	Métodos Construtores e Overloading.....	59
6.4.2	Utilização das API's.....	61
6.4.3	Tipos de Métodos: Públicos, Privados e Protegidos ...	62
6.4.4	Composição e Herança	63
6.4.5	Polimorfismo	66
6.4.6	Métodos Estáticos.....	67
7	Java Server Pages (JSP)	68
7.1	Criando Classes.....	69
7.2	Diretivas	70
7.2.1	Diretiva de Página (page).....	71
7.3	Tags em JSP.....	74
7.3.1	Customização de taglibs.....	75
7.3.2	Tags com conteúdo interno.....	78
7.3.3	Tags com atributos.....	80
8	Servlets e Containers Java	83

8.1	Servlets	84
8.2	Ciclo de Vida e Arquitetura das Servlets.....	85
8.3	Servlets e suas Hierarquias	88
8.4	Mapeamento de Servlets.....	88
9	Acesso ao banco de dados	90
9.1	Utilizando JSP e Javabeans para acessar o MySQL.....	90
9.1.1	Fazendo consultas com JDBC.....	92
	Exercícios	93
	Webliografia	94

CAPÍTULO 3

10	PHP	96
10.1	Sintaxe do PHP.....	99
10.2	Variáveis e tipos	101
10.3	Estruturas de controle	102
10.3.1	Comandos de seleção	102
10.3.2	Comandos de repetição	103
10.4	Array.....	105
10.5	Funções.....	106
10.6	Formulário em PHP.....	107
10.7	Headers	109
10.8	Acesso a Banco de Dados.....	110
10.8.1	Conectando ao banco de dados.....	111
10.8.2	Inserindo dados.....	113
10.8.3	Listando os dados	114
10.8.4	Consultando no banco de dados MySQL.....	116
10.8.5	Excluindo no banco de dados MySQL	117
10.8.6	Alterando no banco de dados MySQL.....	118
10.9	Gerenciando Sessões	119
10.9.1	Manipulando uma sessão.....	119
11	Linguagens de script <i>Client-side</i> - Javascript.....	121
11.1	Linguagens de Script.....	121
11.2	Conceitos Básicos.....	122
11.3	Variáveis	124

11.4	Funções internas.....	125
11.5	Objetos JavaScript.....	126
11.5.1	Button.....	129
11.5.2	Navigator.....	130
11.5.3	Form	131
11.5.4	CheckBox.....	133
11.5.5	Document	136
11.5.6	History.....	139
11.5.7	Window	140
11.5.8	Link	142
	Exercícios	145
	Webliografia	146
	Referências Bibliográficas	147
	Sobre o Autor	149

UNIDADE I

Arquitetura de aplicações WEB

Resumo

O novo paradigma de desenvolvimento de sistemas trouxe novos conceitos aos programadores. Embora a arquitetura cliente-servidor já estivesse estabelecida desde os primórdios da computação, a Internet trouxe novos desafios aos desenvolvedores. Além disso, o HTML se consolidou como a linguagem primordial para as aplicações baseadas na web.

Nesta Unidade examinar-se-á os principais conceitos relacionado ao desenvolvimento web enfatizando a arquitetura cliente-servidor, além de explorarmos o HTML, CSS e XHTML. O texto desta Unidade contempla ideias de vários autores, expressando diversas visões sobre os assuntos.

A Unidade é acompanhada de exercícios sem a solução. Cada questão deve ser encarada como um tema, sobre o qual o aluno deve dissertar. Recomenda-se que seja feita uma pesquisa sobre o assunto e que a questão seja respondida de forma ampla, podendo refletir a opinião do aluno. A bibliografia e a webliografia ao fim dos capítulos e unidades devem ser utilizadas para adquirir um conhecimento razoável sobre o tema de cada capítulo. Ao término da leitura desta Unidade, o estudante deverá: a) Entender o conceito cliente/servidor sobre a ótica do sistemas baseados na web; b) Entender os principais conceitos HTML c) Ser capaz de aplicar o CSS e XHTML no desenvolvimento de páginas web simples.

1. ARQUITETURA DE APLICAÇÕES WEB

1.1 ARQUITETURA CLIENTE/SERVIDOR X SISTEMA BASEADO NA WEB

Um sistema baseado na web diz respeito a um software que se usa através da internet e tendo um *web browser* (Chrome, Firefox, Internet Explorer) como interface. Não é necessário instalar nenhum software extra ou se preocupar com atualizações. Um exemplo é o sistema de *InternetBanking*¹. Outro exemplo são ferramentas de email como *Gmail* ou *Hotmail*. Isso significa que, provavelmente, já se usa sistemas *web based* e talvez não soubesse.

Já um sistema Cliente/Servidor são aqueles que dividem as tarefas para a execução das ações em 2 partes: A parte servidora que é responsável por conectar com o banco de dados e se comunicar com os clientes do sistema através de uma rede de computadores. Normalmente essa rede é uma rede corporativa. A parte cliente se comunica com um ou mais servidores para solicitar algum serviço e aguarda uma resposta deste servidor.

Um sistema baseado na web normalmente é projetado para atender um grande número de usuários, já o cliente-servidor necessita que para ser utilizado o sistema tenha-se um cliente instalado e isso irá limitar o número de acessos. Já um sistema cliente/servidor necessita que o computador cliente possua processamento suficiente para executar o tal programa cliente. O sistema web based necessita que se possa executar um *browser*, aliás muitas vezes não precisa nem ser um computador, pode ser um Tablet ou mesmo um Smart Phone.

O sistema baseado na web necessita de uma internet de qualidade, o que nem sempre é verdade no Brasil, já o sistema cliente-servidor trabalha em uma rede local e normalmente independe da internet.

Nos sistemas baseados na web, os dados trafegam na rede e de algu-

¹ Caracteriza transações, pagamentos e outras operações financeiras e de dados pela Internet por meio de uma página segura de banco.

ma forma podem isso pode não ser muito seguro, se bem que os sistemas de banco mostram o contrario. Já os sistemas cliente-servidor não publicam nenhuma informação na rede e em tese são mais seguros. Os sistemas clientes servidor demandam investimento em pessoal de Tecnologia da Informação, equipamentos, cabeamentos. Já nos sistemas web based, a maior parte do investimento em infra-estrutura fica por conta da empresa fornecedora.

Existem mais pontos que poderíamos comparar, mas com isso pode-se perceber que os sistemas em tese tem uma grande diferença na forma de utilização e com os investimentos em segurança e na computação nas nuvens² os sistemas baseado na web irão expandir rapidamente.

Alguns sistemas, apesar de ser utilizado em uma rede corporativa da empresa e não a internet, é instalado em um equipamento interno da própria empresa e a busca pela informação é restrita ao ambiente corporativo através de *web browser*. Isso para se ter a flexibilidade da internet com a segurança da rede corporativa. As atualizações são feitas de maneira bastante simples e não há a necessidade de se instalar software cliente em cada ponto que será utilizado. Assim como em todos os sistemas baseado na web tradicionais senhas e criptografias são utilizadas para garantir a segurança das informações. Tablets e Smart Phones podem ser utilizados para acessar tais sistemas e finalmente, se for o caso, fica muito simples acessar de fora da empresa.

1.2 SERVIDORES WEB

Servidor web é um programa de computador responsável por aceitar pedidos HTTP³ de clientes, que são geralmente os navegadores, e servi-los com respostas HTTP, incluindo opcionalmente dados, que ge-

2 O conceito de computação em nuvem refere-se à utilização da memória e da capacidade de armazenamento de computadores e servidores compartilhados e interligados por meio da Internet.

3 Protocolo de comunicação (na camada de aplicação segundo o Modelo OSI) utilizado para sistemas de informação de hipermídia, distribuídos e colaborativos. Ele é a base para a comunicação de dados da World Wide Web.

almente são páginas web, tais como documentos HTML com objetos embutidos (imagens, etc.) ou um computador que executa um programa que provê a funcionalidade descrita anteriormente. O mais popular, e mais utilizado no mundo, é o servidor Apache⁴ (software livre). A Microsoft possui a sua própria solução denominada IIS (Internet Information Services).

Uma comunicação simples entre o cliente e o servidor Web funciona da seguinte forma: o browser do cliente decompõe a URL (o endereço da página) em várias partes, tais como o nome de domínio, nome da página e protocolo. Por exemplo, para a URL <http://www.ufpi.br/calendario.php>, o protocolo é o *http*, o nome de domínio é *ufpi.br* e o nome da página é *calendario.php*.

Um Servidor de nome de Domínio (DNS – *Domain Name System*) traduz o nome de domínio, informado pelo usuário, para seu endereço de IP, que é uma combinação numérica que representa o endereço real do site na internet. Por exemplo, o domínio *ufpi.br* é traduzido para 210.132.233.92. O browser agora determina qual protocolo deve ser usado. Os exemplos de protocolos incluem FTP (Protocolo de Transferência de Arquivos) e *http* (Protocolo de Transferência de HiperTexto).

O servidor passa então a recuperar os arquivos solicitados na página. Por exemplo, quando um usuário digitar *http://http://portal.mec.gov.br/index.php*, o browser solicita o arquivo *index.php* do servidor *portal.mec.gov.br* e aguarda uma resposta. O servidor então responde os pedidos do browser: verifica se o endereço existe, encontra os arquivos necessários, executa as instruções apropriadas e retorna os resultados. Se não puder localizar o arquivo, o servidor retorna uma mensagem de erro para o cliente. O browser recebe os dados do servidor na linguagem HTML, interpreta essas instruções e exhibe os resultados para o usuário. Esse processo é repetido até que o cliente que utilize o browser deixe o site.

Considerando que um *browser* simplesmente traduz e exhibe os dados alimentados, um servidor Web é responsável por distinguir os vários

4 <http://www.apache.org/>

tipos de erros e dados. Deve, por exemplo, designar o código apropriado para qualquer erro interno e enviar ao browser logo a informação. Deve ainda distinguir vários elementos de uma página Web (como .GIFs, .JPEGs , entre outros), de forma que o browser saiba quais arquivos usar na hora de formatar a página.

Dependendo da função do site, um servidor Web pode também tratar de tarefas adicionais, como registro de estatísticas, segurança de manipulação e criptografia, servir imagens para outros sites, gerenciador de conteúdo dinâmico, ou funções de comércio eletrônico.

1.3 CLIENTES WEB

Após vários modelos estudados de cliente-servidor caracterizou-se chamar tecnicamente de arquitetura multicamada, inspirado nas camadas no Modelo OSI5, o processo de dividir a arquitetura de cliente-servidor em várias camadas lógicas facilitando o processo de programação distribuída. Existe desde o modelo mais simples de duas camadas, e o mais utilizado atualmente, MVC, que é o modelo de três camadas (capítulo 5). São características dos clientes web:

- Inicia pedidos para servidores;
- Espera por respostas;
- Recebe respostas;
- Conecta-se a um pequeno número de servidores de uma só vez ;
- Normalmente interage diretamente com os servidores através de seu software aplicação específico, que lhe possibilita a comunicação com o servidor;
- Utiliza recursos da rede.

2. HTML

5 Modelo de rede de computador referência da ISO dividido em camadas de funções, com objetivo de ser um padrão, para protocolos de comunicação entre os mais diversos sistemas em uma rede local, garantindo a comunicação entre dois sistemas computacionais

Os documentos disponíveis na internet, independentemente da temática que abordem (notícias, entretenimento, ciência, comércio, etc.), são estruturados através de uma Linguagem de Marcação de Hipertexto conhecida como HTML (*Hypertext Markup Language*). Uma linguagem de marcação é um mecanismo para adicionar marcas com algum significado a um texto. Tais marcas são omitidas na versão do texto que é apresentada ao usuário (NIEDERST, 2002).

O HTML é também conhecido como uma linguagem de formatação e não de programação (BRITO, 2011). Essa linguagem tem a função de enviar para o navegador (Internet Explorer, Firefox, Chrome, etc.) informações que definem de que maneira textos, imagens e outros itens deverão aparecer na tela. Essas informações são chamadas *tags* (etiquetas) e estão inseridas nos documentos originais (documento-fonte) que criaram as páginas. Essas tags normalmente aparecem em pares, iniciando e encerrando um bloco. Existem quatro tags básicas são elas:

```
<html></html>, <head></head>, <title></title>, <body></body>
```

Para se obter um documento escrito em HTML é necessária a utilização do editor de texto e de, no mínimo, essas quatro tags.

`<html>... </html>`: abre e encerra uma página. Essas tags são as mais importantes, pois informam ao browser que o documento está escrito em html.

`<head> ...</head>`: a tag `<head>` vem abaixo da tag `<html>`, ela indica os comandos que o navegador precisa carregar antes que a página seja carregada. É utilizada no cabeçalho para inibir o título a ser inserido na janela do navegador.

`<title>... </title>`: espaço para definição do título do documento. A tag `<title>` deve estar sempre contida na tag `<head>`.

`<body></body>`: essa tag contém o corpo da página. A tag `<body>` deve ser inserida após a tag `<head>`; já a tag `</body>` vem antes da tag `</html>`.

A especificação atual da linguagem HTML é a versão 5 e é padro-

nizada pelo consórcio W3C (BONIATI, 2013). Contudo, ao longo dos nossos exemplos, procuraremos utilizar instruções compatíveis entre as diferentes versões. Não perca de vista, entretanto, os avanços e os recursos disponíveis a partir de novas versões. É importante estar sempre atualizado e disponível para aprender.

2.1 MARCADORES

Como já mencionado, os marcadores são comandos da linguagem HTML que permitem a formatação do texto. Um marcador deve ser apresentado entre os sinais “<” e “>”. A maioria dos marcadores funciona como chave de liga e desliga. Isso quer dizer que um marcador é utilizado para indicar o início da formatação e outro para informar o fim dela; no caso do fechamento, é inserido “/”, antes do nome do marcador (BRITO, 2011).

2.1.1 Cabeçalho

As tags <h1>, <h2>, <h3>, <h4>, <h5> e <h6> são utilizadas para demarcar uma área do documento que indica um cabeçalho – head (um título ou subtítulo, por exemplo). Quanto menor for o valor, mais destaque receberá a apresentação do cabeçalho (BONIATI, 2013). Os cabeçalhos são exibidos em negrito e, ao final dos mesmos, é feita uma quebra de linha. A utilização de <h1> </h1>, por exemplo, demarcam uma área do texto que merece realce e, normalmente, são utilizados para informar, o título do mesmo. Opcionalmente, pode-se utilizar o parâmetro align para indicar o alinhamento do cabeçalho: right (a direita), left (a esquerda) ou center (centralizado).

2.1.2 Parágrafos

A tag `<p>` (paragraph) demarca um parágrafo textual. Dividir um texto em parágrafos é uma atividade presente em qualquer redação. Associado à tag `<p>`, existe um parâmetro denominado de `align` o qual informa o alinhamento do texto, podendo este ser centralizado (`center`), justificado (`justify`), alinhado à esquerda (`left`) ou alinhado à direita (`right`). A utilização da tag `<p>` de forma vazia, como `<p></p>`, produz uma quebra de linha. Todavia, existe uma tag especial para esse propósito, conforme veremos a seguir.

2.1.3 Textos pré-formatados

Há situações nas quais temos a necessidade de manter a apresentação de um texto tal como ele foi digitado (ou seja, respeitando espaços, tabulações e quebras de linha). Uma típica situação em que isto acontece é quando precisamos representar um exemplo de linguagem de programação. Nesse caso, a opção indicada é a tag `<pre>` (predefined). Seu efeito visual é um recuo à esquerda e a apresentação do texto em uma fonte de tamanho fixo tal como a usada quando este foi digitado (respeitando quebras de linha, espaços e tabulações).

Observe, na Figura 1, o desenho de uma tabela apenas com sinais gráficos. Veja que, com a tag `<pre>`, no momento em que o código é visualizado, ele mantém as posições do texto tal como foram predefinidas no documento HTML.

```

<html>
  <body>
    <pre>
      Product Name  Qty  Price  Total
      =====
      Product1     12   100   1200
      Product2     10   200   2000
      Product3     15   150   2250
      =====
      Total                               5450
      =====
    </pre>
  </body>
</html>

```


Product Name	Qty	Price	Total
Product1	12	100	1200
Product2	10	200	2000
Product3	15	150	2250
Total			5450

2.1.4 Quebras de linha

Figura 1: Exemplo de uso da tag <pre>

Conforme discutimos anteriormente, o navegador ignora as quebras de linha informadas ao longo do texto. Para produzir o efeito desejado, devemos utilizar a tag
 (break), que é independente, ou seja, não precisa de outra tag para fechá-la, bastando, para isto, indicar, na própria instrução de abertura, o sinal de “/” para fechá-la automaticamente.

A tag <hr /> (head row) tem o mesmo princípio da tag
, no entanto seu efeito é a produção de uma linha que divide a página horizontalmente. Seu intuito é produzir seções de divisão ao longo do conteúdo.

2.1.5 Fonte

A tag atua sobre atributos do texto em si. Em síntese, são três os atributos que podemos alterar: o tamanho (size), o tipo (face) e a cor (color). O tamanho é definido a partir da utilização de um número inteiro e pode ser empregado de forma absoluta (informando diretamente o número) ou de forma relativa, indicando, através dos sinais positivo (+) ou negativo (-), o valor de incremento ou decremento em relação ao valor padrão do navegador (MARCONDES, 2005). O valor absoluto para o atributo size (que define o tamanho) pode variar entre 1 e 7 – quanto maior o número, maior será o tamanho de sua exibição.

A alteração do tipo de fonte (da tipografia ou do formato das letras) pode ser feita com o atributo `face`. Nesse caso, é importante ressaltar que nem todos os usuários, dispositivos ou plataformas compartilham das mesmas fontes. Dessa forma, procure utilizar tipos de fontes altamente difundidas (ex.: Arial, Times New Roman, Verdana). O atributo `face` permite especificar mais de um tipo de fonte. Assim, se o primeiro não existir, usa-se o segundo assim por diante. Para indicar mais de uma fonte, os nomes destas devem ser separados por vírgulas.

Por fim, o atributo `color` permite alterar a cor de exibição de uma fonte. Sua utilização é bastante simples, contudo precisamos entender de que forma o HTML nomeia ou utiliza as cores. A representação das cores em HTML utiliza do modelo de cores RGB (Red/Green/Blue), cujo propósito é a reprodução de cores em dispositivos eletrônicos. A partir das cores básicas do modelo (vermelho, verde e azul), são feitas combinações com diferentes intensidades de cada uma das cores, conseguindo uma variedade de 16 milhões de cores (BONIATI, 2011).

As combinações mais básicas podem ser expressas através de nomes, no entanto a forma mais interessante de se conseguir uma cor pelo sistema RGB é misturar os tons. Cada tom pode variar de 0 a 255, o que pode ser representado por dois dígitos hexadecimais. A cor cujo código é `#FF0000` e representa, por exemplo, o vermelho puro, em função de que os dois primeiros dígitos estão com o valor hexadecimal mais alto possível. Já a cor de código `#000000` representa o preto (a ausência total de cor, em qualquer tonalidade). Observe, na Figura 2, a relação dos nomes de cores reconhecidos pela W3C⁶ (World Wide Web Consortium). Além destes, é possível variar os valores de cada dupla de dígitos hexadecimais e conseguir 16.777.216 combinações (256³).

⁶ World Wide Web Consortium (W3C) é a principal organização de padronização da World Wide Web. Consiste em um consórcio internacional com quase 400 membros[1], agrega empresas, órgãos governamentais e organizações independentes com a finalidade de estabelecer padrões para a criação e a interpretação de conteúdos para a Web.

Figura 2: Quadro de cores básicas W3C. O quadro apresenta uma lista de 12 cores com seus respectivos nomes e códigos HEX. Cada linha contém o nome da cor, o código hexadecimal e uma pequena amostra da cor em um retângulo.

Color Name	Color HEX	Color
aqua	#00FFFF	
black	#000000	
blue	#0000FF	
fuchsia	#FF00FF	
gray	#808080	
green	#008000	
lime	#00FF00	
maroon	#800000	
navy	#000080	
olive	#808000	
purple	#800080	
red	#FF0000	
silver	#C0C0C0	
teal	#008080	

Além de alterar o tipo de fonte, podemos marcar áreas de texto que precisam ser enfatizadas. Estilos físicos, como negrito, itálico e subscrito, podem ser conseguidos com tags específicas. A Tabela 1 demonstra os principais recursos que podemos utilizar para enfatizar blocos de conteúdo textual.

Tabela 1: Ênfase em blocos textuais

Tag	Propósito	Exemplo	Resultado
 (bold)	Negrito	escuro	escuro
<i> (italic)	Itálico	<i>deitado</i>	<i>deitado</i>
<u> (underline)	Sublinhado	<u>com linha</u>	<u>com linha</u>
<s> (strike)	Riscado	<s>tachado</s>	tachado
<tt> (teletype)	Espaçamento fixo	<tt>fixo</tt>	fixo
<sub> (subscript)	Subscrito	H₂O	H ₂ O
<sup> (superscript)	Sobrescrito	km²	km ²

2.1.6 Caracteres especiais

Imagine uma situação na qual necessitamos separar um texto por um conjunto de espaços em branco. É sabido que espaços em branco, em sequência, são ignorados pelo navegador. Então, como conseguir representá-los? Imagine gora uma situação em que precisamos representar o sinal de micro (μ) ou algum outro sinal não encontrado, normalmente, no teclado. Essas situações podem ser resolvidas com a utilização de caracteres especiais, também conhecidos como caracteres ISO.

O alfabeto de caracteres ISO (*International Organization for Standardization*) é obtido por um código especial formado pelos caracteres & (“e” comercial), #(tralha ou sustenido), a definição de um valor numérico de três dígitos, o caractere“;” (ponto-e-vírgula) para finalizar. É possível, ainda, utilizar uma espécie de abreviação ou entidade equivalente ao código ISO. Por exemplo, para representar o sinal de “menor que” podemos utilizar o código < ou então a entidade< onde lt é a abreviação para letter then (MANZANO; TOLEDO, 2008).

A utilização dos códigos do alfabeto de caracteres ISO, por meio de entidades, garante que o conteúdo do documento HTML será exibido da forma como foi definido, independente do navegador ou da plataforma utilizada.

Pelas limitações de tamanho e de uso em ambientes multilinguais, O Unicode Consortium desenvolveu o padrão unicode. Este cobre todos os caracteres, pontuações e símbolos do mundo. O Unicode habilita o processamento, armazenamento e intercâmbio de dados de texto independentemente de plataforma, programa ou linguagem. A Tabela 2 apresenta os caracteres do alfabeto ISO e o nome da entidade correspondente.

Tabela 2: Caracteres ISO

Á	Á	Í	Í	Ú	Ú
á	á	í	í	ú	. ú
Â	Â	Î	Î	Û	.. Û
â	â	î	î	û	.. û
À	À	Ï	Ì	Û Ù
à	à	ï	ì	ù	ù
Å	Å	ì	ì	Ü	... Ü
å	å	ï	ï	ü	... ü
Ã	Ã	Ï	Ï	Ç	. Ç
ã	ã	ï	ï	ç	. ç
Ä	Ä	Ó Ó	Ñ	. Ñ
ä	ä	ó	ó	ñ	. ñ
Æ	Æ	Ô	.. Ô	<	... <
æ	... æ	ô	... ô	>	... >
É	.. É	Ò	Ò	&	.. &
é	. é	ò	. ò	"	... "
Ê	... Ê	Ø	.. Ø	® ®
ê	.. ê	ø	. ø	©	. ©
È	.. È	Õ	.. Õ	Ý Ý
è	. è	õ	. õ	ý ý
Ë Ë	Ö Ö	Þ	Þ
ë	... ë	ö	... ö	þ	.. þ
Ð Ð			ß	.. ß
ð ð				

2.2 FORMULÁRIOS

Formulários HTML são um dos principais pontos de interação entre um usuário e um web site ou aplicativo. Eles permitem que os usuários enviem dados para o web site. Na maior parte do tempo, os dados são enviados para o servidor da web, mas a página da web também pode interceptar para usá-los por conta própria.

Um formulário HTML é feito de um ou mais *widgets*. Esses *widgets*

podem ser campos de texto (linha única ou de várias linhas), caixas de seleção, botões, checkboxes ou *radio buttons*. A maior parte do tempo, estes elementos são emparelhados com uma legenda que descreve o seu objetivo.

A principal diferença entre um formulário de HTML e um documento regular de HTML é que, maioria das vezes, o dado coletado é enviado ao servidor. Nesse caso, você precisa configurar um servidor web para receber e processar os dados. Mais adiante, nos capítulos 6 (Java) e 10 (PHP), veremos com estabelecer conexões com banco de dados de forma a guardar as informações oriundas dos formulários.

2.2.1 O Elemento `<form>`

Todos formulários HTML começam com um elemento `<form>` como este:

```
<form action="/pagina-processa-dados-do-form" method="post">  
    ....  
</form>
```

Este elemento define um formulário. É um elemento de container como um elemento `<div>` ou `<p>`, mas ele também suporta alguns atributos específicos para configurar a forma como o formulário se comporta. Todos os seus atributos são opcionais, mas é considerada a melhor prática sempre definir pelo menos o atributo `action` e o atributo `method`. O atributo `action` define o local (uma URL) em que os dados recolhidos do formulário devem ser enviados. O atributo `method` define qual o método HTTP para enviar os dados. Ele pode ser “GET” ou “POST”. Mais adiante, no capítulo 10, veremos a diferença entre eles.

2.2.2 Elementos `<label>`, `<input>`, e `<textarea>`

Criaremos, a título de exemplo, três campos de texto cada um com uma etiqueta. O campo de entrada para o nome será um campo básico texto de linha única (“input”); o campo de entrada do e-mail será um campo de texto com uma única linha (“input”) que vai aceitar apenas um endereço de e-mail; o campo de entrada para a mensagem será um campo de texto de várias linhas (“textarea”). Em termos de código HTML, teremos algo assim:

```
<form action="/pagina-processa-dados-do-form" method="post">
<div>
<label for="nome">Nome:</label>
<input type="text" id="nome" />
</div>
<div>
<label for="email">E-mail:</label>
<input type="email" id="email" />
</div>
<div>
<label for="msg">Mensagem:</label>
<textarea id="msg"></textarea>
</div>
</form>
```

Os elementos <div> estão lá para estruturar nosso código e deixar a estilização mais fácil. Observe o uso do atributo em todos os elementos <label>; é uma maneira para vincular uma label a um campo do formulário. Este atributo faz referência ao id do campo correspondente. Há algum benefício para fazer isso, é o de permitir que o usuário clique no rótulo para ativar o campo correspondente.

No elemento <input>, o atributo mais importante é o atributo

type. Esse atributo é extremamente importante porque define a forma como o elemento `<input>` se comporta. Ele pode mudar radicalmente o elemento, então preste atenção a ele. Se você quiser saber mais sobre isso, leia o artigo *native form widgets*. No exemplo anterior, usamos somente o `type="text"`, valor padrão para este atributo. Ele representa um campo de texto com uma única linha que aceita qualquer tipo de texto sem controle ou validação. Usamos o `type="email"` que define um campo de texto com uma única linha que só aceita um endereço de e-mail bem-formatados. Este último valor torna um campo de texto básico em uma espécie de campo “inteligente”, que irá realizar alguns testes com os dados digitados pelo usuário. Para saber mais sobre a validação de formulário, detalharemos melhor no artigo *Form data validation*.

Por último, mas não menos importante, observe a sintaxe de `<input />` e `<textarea></ textarea>`. Esta é uma das esquisitices do HTML. A tag `<input />` é um elemento que se auto-fecha, o que significa que se quiser encerrar formalmente o elemento, você tem que adicionar uma barra “/” no final do próprio elemento e não uma tag de fechamento. No entanto, o tipo `<textarea>` não é um elemento de auto-fechamento, então é necessário fechá-lo com a tag final adequada. Isso tem um impacto sobre um recurso específico de formulários HTML: a maneira como você define o valor padrão. Para definir o valor padrão de um elemento `<input>` você tem que usar o atributo `value` como este:

```
<input type="text" value="Por padrão, este elemento será preenchido com este texto" />
```

Pelo contrário, se desejamos definir o valor padrão de um elemento `<textarea>`, é necessário colocar esse valor padrão no meio das tags, entre tag inicial e a tag final do elemento `<textarea>`, como abaixo:

```
<textarea>Por padrão, este elemento será preenchido com este texto </textarea>
```

2.2.3 Elemento `<button>`

Para finalizarmos o formulário temos apenas que adicionar um botão para permitir que o usuário envie seus dados depois de ter preenchido o formulário. Isto é simplesmente feito usando o elemento `<button>`:

```
<form action="/pagina-processa-dados-do-form" method="post">
<div>
<label for="name">Nome:</label>
<input type="text" id="name" />
</div>
<div>
<label for="mail">E-mail:</label>
<input type="email" id="mail" />
</div>
<div>
<label for="msg">Mensagem:</label>
<textarea id="msg"></textarea>
</div>
<div class="button">
<button type="submit">Enviar sua mensagem</button>
</div>
</form>
```

Um botão pode ser de três tipos: `submit`, `reset`, ou `button`. Um botão de `submit` envia os dados do formulário para a página de web definida pelo atributo `action` do elemento `<form>`. Já o botão de `reset` redefine imediatamente todos os campos do formulário para o seu valor padrão. Por fim, um tipo *button* faz nenhuma ação por si só. Isso soa bobo, mas é incrivelmente útil para construir botões personalizados com JavaScript (Capítulo 11), ou seja, ele pode assumir qualquer comportamento através desta linguagem.

Note que também pode-se usar o elemento `<input>` com o tipo

correspondente para produzir um botão. A principal diferença com o elemento `<button>` é que o elemento `<input>` permite apenas texto sem formatação como seu valor, enquanto que o elemento `<button>` permite que o conteúdo HTML completo como seu valor.

3. CSS

O CSS (Cascading Style Sheets – Folhas de Estilo em Cascata) é uma linguagem de estilo que foi desenvolvida para controlar cores, margens, fontes, linhas, alturas, larguras, imagens de fundo, entre outros (BRITO, 2011). O HTML tem algumas tags com essas funções, porém nem sempre serão suficientes para suprir a necessidade de encontrar meios de construir layouts para os documentos online.

Para suprir a necessidade de novas possibilidades de criação de layouts foi criado o CSS. Dessa forma, o HTML passou a ser utilizado apenas como linguagem de marcação e estruturação, enquanto o CSS a função de aplicar estilos necessários para a aparência da página desenvolvida.

As CSS têm por finalidade devolver ao HTML o propósito inicial da linguagem, ou seja, a marcação e a estruturação de conteúdos. Ao HTML não cabe fornecer informações ao navegador sobre a apresentação dos elementos – cores de fontes, tamanhos de textos, posicionamento e todo o aspecto visual de um documento não devem ser funções do HTML. Todas as funções de apresentação cabem a CSS, sendo esta a sua finalidade maior. HTML para estruturar e CSS para apresentar (SILVA, 2007).

Com a criação do CSS, houve alguns benefícios para o desenvolvimento de websites, entre eles: a precisão no controle do layout, a criação da folha de estilos, possibilitando o controle de vários documentos a partir de um, a possibilidade de criar layouts específicos para determinadas mídias, telões e dispositivos móveis, entre outros.

Para aplicação do CSS são criadas folhas de estilos, documento com

extensão CSS que conterà os códigos de definição de estilo de determinado documento, que pode ser de extensão HTML (BONIATI, 2013). Essa folha pode ser um documento separado, contendo apenas os códigos de estilos, vinculado ao arquivo HTML ou, então, esses códigos podem ser digitados diretamente no arquivo HTML.

Os estilos definidos pelo CSS são aplicados conforme a seguinte sintaxe:

```
elemento {atributo1: valor; atributo2: valor...}
```

Podemos descrever cada item dessa síntese como:

- Elemento: descreve o elemento de design ao qual o estilo será aplicado. Essa é a mesma tag HTML, mas sem os sinais de maior e menor. Essa parte da regra é, às vezes, chamada de selector.
- Atributo: aspecto específico do elemento que se quer usar como estilo. Deve ser um nome de atributo CSS válido, como o atributo font-size.
- Valor: esse item configura a aplicação do atributo. Deve ser uma configuração válida para o atributo em questão, como 20 pt (20 pontos) para font-size.
- Atributo valor: nesse item da síntese pode-se atribuir múltiplas declarações que podem ser separadas com ponto e vírgula (;). Porém, no último item não coloque ponto e vírgula.
-

Veja um exemplo de regra que diz que todos os títulos de nível 2 (tags <H2>) devem ter tamanho de 24 pontos e cor azul:

```
H2 {font-size: 24pt; color: blue}
```

Para que fique claro e visível que se colocou todos os sinais de ponto e vírgula e chavetas nos lugares corretos, utilize quebras de linha e espaços em branco na regra. Exemplo:

```
P {font-family: Times;
```

```

    font-size: 12pt;
    color: blue;
    margin-left: 0.5in}

```

O exemplo acima descreve que os parágrafos deverão aparecer em fonte Times, 12 pontos, azul, recuada meia polegada, a partir da margem esquerda da página.

3.1 MÉTODO DE APLICAÇÃO DO CSS NO DOCUMENTO HTML

Nesta sessão veremos os principais métodos de aplicação do CSS em um documento HTML: métodos inline, interno e externo.

3.1.1 MÉTODO INLINE

É aplicado usando o atributo *style* do HTML, acrescenta-se dentro do atributo *style* a característica que se deseja obter naquele documento. Com esse método, é possível conseguir poucos efeitos. Os estilos criados por esse método só afetam a tag na qual ele está inserido, não afeta outras tags e nem mesmo outros documentos (BONIATI, 2013).

Ao utilizar a tag *style*, não é necessário utilizar os colchetes, nem acrescentar a tag `</style>` de fechamento. No CSS as regras devem ser colocadas entre aspas, separando-as com o ponto e vírgula (Figura 3).

```

1 <html>
2   <head>
3     <title>Exemplo</title>
4   </head>
5   <body style="background-color: #008000;">
6
7     <p>Esta é uma página com fundo verde</p>
8   </body>
9 </html>

```

Figura 3: Método inline (BRITO, 2011)

3.1.2 Método interno

Esse método também é aplicado utilizando a tag `<style>` do HTML; porém nele a tag é colocada na própria página HTML (Figura 4), ao invés de separado como no método inline.

```

1  <html>
2  <head>
3      <title>Exemplo</title>
4      <style type="text/css">
5
6          body {background-color: #008000;}
7      </style>
8  </head>
9  <body>
10     <p>Esta é uma página com fundo verde</p>
11
12 </body>
13 </html>

```

Figura 4: Método Interno (BRITO, 2011)

3.1.3 Método externo

Nesse método é criado um arquivo separado com os estilos. Nesse método é criado um arquivo separado com os estilos, que deve ser salvo com a extensão CSS e salvá-lo no mesmo diretório que o arquivo em HTML, como mostra o exemplo na Figura 5.



Figura 5: Método Externo

Para aplicar os estilos em uma nova página HTML, basta inserir uma tag `<link>` no cabeçalho que faça referência ao arquivo .CSS, conforme a Figura 6. Note que o caminho para a folha de estilos é indicado no atributo href.

```
1  <html>
2  <head>
3    <title>Meu documento</title>
4    <link rel="stylesheet" type="text/css" href="style/style.css" />
5
6  </head>
7  <body>
8  ...
```

Figura 6: Exemplo do método externo (BRITO 2011)

3.2 PSEUDOCASSES E PSEUDOELEMENTOS

A utilização de regras CSS não está restrita aos seletores do tipo tags HTML. Existem três tipos de seletores que merecem nossa atenção: classes (*class*), identificadores (*id*) e pseudoclasses. Um seletor do tipo classe permite-nos alguns recursos interessantes. Por meio dele, podemos aplicar um mesmo estilo a diferentes elementos e também indicar que um mesmo elemento obedece a diferentes estilos.

Para definir um seletor do tipo *class*, devemos iniciar a especificação do mesmo com o sinal de “.” (ponto final). A utilização de tal seletor, em elementos do documento HTML, ocorre por meio do atributo *class* que é aceito pela maioria das tags HTML.

Um seletor do tipo identificador é utilizado com o atributo *id* o qual identifica, de forma única e exclusiva, um determinado elemento no documento HTML. A declaração de um seletor do tipo *id* inicia-se com o sinal de “#” (tralha ou sustenido) seguido do seu nome. A especificidade de um seletor *id* é maior do que a de um seletor *class*. Dessa forma, se em um mesmo elemento HTML forem utilizados os dois atributos, serão aplicadas as regras de estilo do identificador.

Há também um tipo especial de seletor denominado de pseudoclas- se, que é utilizado para alcançar elementos que não são especi cados por meio de uma marca. Por exemplo, poderíamos aplicar efeitos diferentes em links ainda não visitados, já visitados ou aquele em que o usuário está

com o cursor do mouse em cima. Para isso podemos usar as pseudoclasses. Observe que duas pseudoclasses estão sendo utilizadas para oferecer efeitos diferenciados para a tag `<a>`: *visited* (que permite estilizar um link já visitado) e *hover* (que permite estilizar um link no momento em que o cursor do mouse está sobre ele).

3.3 PROPRIEDADES

As propriedades, conforme já estudamos, formam, em conjunto com seus valores, a unidade mais básica de definição de uma regra. Propriedades são como palavras reservadas da sintaxe CSS e são predefinidas (MARCONDES, 2005). A lista de propriedades da linguagem é bastante extensa; vejamos algumas:

- `font-family` – nome da fonte (o tipo de letra).
- `font-style` – estilo itálico (`italic`) ou oblíquo (`oblique`).
- `font-weight` – largura da fonte (ex.: negrito – `bold`).
- `text-decoration` – especifica uma decoração para o texto, como sublinhado (`underline`), riscado (`line-through`), sublinhado invertido (`overline`).
- `text-transform` – define-se o texto será apresentado em letras maiúsculas (`uppercase`), minúsculas (`lowercase`), com as iniciais em maiúsculas (`capitalize`).
- `font-size` – especifica o tamanho da fonte (veja mais sobre unidades de medida na seção seguinte).
- `letter-spacing` – espaçamento entre letras.
- `word-spacing` – espaçamento entre palavras.
- `color` – cor do texto. Pode ser especificada em RGB hexadecimal (ex.: `#FF00CC`), RGB decimal (ex.: `RGB [255, 0, 100]`), RGB percentual (ex.: `RGB [100%, 0%, 50%]`) ou por meio do nome da cor (ex.: `red`).

- background-color – cor de fundo.
- background-image – imagem de fundo. Em CSS, para **especificar** um arquivo externo, como uma imagem, utilizamos a função URL (ex.: URL “./imagens/fundo.jpg”).
- background-repeat – define como a imagem de fundo será repetida: repeat (repete tanto verticalmente quanto horizontalmente), no-repeat (exibe a imagem apenas uma vez), repeat-x (repetição apenas no eixo x – horizontal), repeat-y (repetição apenas no eixo y – vertical).
- background-attachment – determina o comportamento da imagem em relação à rolagem da tela (página ou camada). Pode ser: fixed (a imagem não acompanha a rolagem) ou scroll (a imagem movimenta-se de acordo com a rolagem da tela).
- background-position – posicionamento da apresentação da imagem. Essa propriedade precisa de dois valores: o primeiro para identificar o posicionamento vertical (top, bottom e center) e o segundo para identificar o posicionamento horizontal (left, right e center).
- text-align – alinhamento do texto: left (à esquerda), right (à direita), center (centralizado) e justify (justificado).
- text-indent – recuo de primeira linha.
- line-height – espaçamento entre linhas.
- border-width – espessura da borda: thin (fina), medium (média), thick (grossa). Também, é possível acessar propriedades **específicas** de cada uma das bordas. Ex.: border-left-width – altera a espessura da borda da esquerda (outras opções são: border-right, border-top e border-bottom).
- border-color – cor da borda.

- `border-style` – estilo ou decoração: `hidden`, (oculta), `solid` (linha simples sem efeito), `ridge` (3D), `double` (dupla), `dotted` (linha pontilhada), `dashed` (linha tracejada), `outlet` (3D em alto-relevo), `inset` (3D em baixo relevo), `none` (invisível).

3.4 UNIDADES DE MEDIDA

As propriedades que listamos, na seção anterior, podem ser qualificadas (quando seu valor é uma informação textual) ou quantificadas (quando seu valor é, por exemplo, algo mensurável ou algo que se utiliza de valores). O tamanho de uma fonte, a espessura de uma borda ou as margens de um parágrafo são exemplos de propriedades que podem ser quantificadas. O CSS trabalha com diferentes unidades de medida. É importante que as conheçamos para ajustar os valores de nossas propriedades de forma adequada (BONIATI, 2013).

As unidades de medida utilizadas em CSS se dividem em absolutas e relativas. As unidades absolutas são aquelas que não dependem de nenhum outro valor de referência, ou seja, são definitivas (SILVA, 2007).

- `in` – polegada.
- `cm` – centímetro.
- `mm` – milímetro.
- `pt` – ponto (1 pt = 1/72 polegadas).
- `pc` – pica (1 pc = 12 pt).

Unidades de medidas relativas são aquelas que se utilizam de um valor de referência anteriormente definido. São de três tipos:

- `em` – calculada em relação ao tamanho de fonte predefinido (1 em é igual ao tamanho da fonte definido para o elemento a ser estilizado).

- ex – calculada em função da altura da letra xis (x) minúscula da fonte adotada.
- px – refere-se a um pixel. No entanto, como diferentes dispositivos utilizam diferentes resoluções de tela, o tamanho do pixel pode variar de dispositivo para dispositivo.

4 XHTML

Quando o W3C definiu os parâmetros da quarta versão do HTML (*Hyper Text Markup Language*) em 1997, os profissionais ficaram satisfeitos com o resultado final e o adotaram sem problemas como a base definitiva de programação para a criação de páginas na Web. Depois de três anos, a linguagem sofre o que parece ser sua maior evolução com a adição de aplicações da metalinguagem XML (*EXtensible Markup Language*). Nasce, então o XHTML 1.0, a nova linguagem-base para criação de páginas Web que reúne todas as qualidades do HTML com os recursos do XML, destinado para substituir, aos poucos, o HyperText 4.0 (GONÇALVES, 2005).

Todas as linguagens de marcação da web são baseadas em SGML⁷, uma metalinguagem complexa projetada para máquinas com a finalidade de servir de base para criação de outras linguagens.

O SGML foi usado para criar XML (*Extensible Markup Language*), também uma metalinguagem, porém bem mais simples. Com XML cria-se suas próprias tags e atributos para escrever seu documento web. Isto significa que é o próprio usuário quem cria sua linguagem de marcação. XHTML foi criado dentro deste conceito e por isso é uma aplicação XML. As tags e atributos do XHTML foram criadas (“inventadas”) aproveitando-se as nossas conhecidas tags e atributos do HTML 4.01 e suas regras.

7 Metalinguagem através da qual se pode definir linguagens de marcação para documentos.

XHTML é uma linguagem de marcação bastante familiar para quem conhece HTML e a transformação de um documento existente de HTML para XHTML é simples. A junção das duas linguagens resultou no XHTML (*EXtensible HyperText Markup Language*), uma linguagem quase igual ao HTML original, o que facilita muito aos programadores que estão acostumados com todas as tags e códigos desde que foi criada, mas que é capaz de apresentar a “flexibilidade” da linguagem XML de levar seu conteúdo registrado nela para outras plataformas.

Graças à proximidade do XHTML 1.0 com seu antecessor, o HTML 4.0, os elementos XML podem ser inseridos nas páginas HTML já existentes, adicionando as novas tags e elementos originados da nova linguagem, gerando infinitas novas possibilidades para o futuro da Web em termos de divulgação, de conteúdo e de aperfeiçoamento da programação.

A principal vantagem da XHTML é a compatibilidade da linguagem com as futuras aplicações de usuários, garantindo desde já que as criações XHTML se consevarão estáveis por longos anos. A tendência é que futuras versões de browsers e agentes de usuários em geral, deixem de suportar elementos e atributos já em desuso (“*deprecated*”) segundo as recomendações da W3C, assim como antigos e ultrapassados esquemas e esboços do HTML (GONÇALVES, 2005).

Um arquivo XHTML é um arquivo HTML, que pode ser lido por qualquer browser. Não estamos falando de um novo HTML, com novas tags ou coisa assim. Pelo contrário, o XHTML foi feito para funcionar mesmo em navegadores antigos. Mas, ao mesmo tempo, Um arquivo XHTML é também um arquivo XML válido, que pode ser lido por qualquer interpretador de XML.

4.1 CONCEITOS BÁSICOS

Para que seu arquivo possa ser lido por máquinas além de humanos, é muito importante que se escreva um XHTML válido, com isso faze-se

com que as informações do seu site fiquem mais acessíveis para as buscas, contribuindo para o projeto e principalmente melhorando seu site.

Todos os documentos XHTML podem ser divididos em duas partes básicas: o cabeçalho e o corpo, assim como duas especiais: a declaração da versão e a declaração do tipo do documento. O cabeçalho possui informações sobre o documento em si, seus principais componentes são (Tabela 2):

Tabela 3: Declarações XHTML

Elemento	Descrição	Obrigatório
<code><title></title></code>	Deve ser o primeiro elemento do cabeçalho, informa o texto que aparece na barra de título do navegador	Sim
<code><meta></meta></code>	Contém informações sobre o conteúdo do documento	Não
<code><link></link></code>	Elemento utilizado para realizar a ligação entre os documentos e as páginas contendo as folhas de estilo	Não
<code><style></style></code>	Informações sobre as folhas de estilo usadas em um determinado documento	Não
<code><object></object></code> <code><script></script></code>	Espaço utilizado para inserção de código, por exemplo: java script	Não
<code><base></base></code>	Especifica o endereço do documento XHTML	Não

A declaração de versão, que informa versão da linguagem XML que será utilizada na descrição do documento, também faz parte do cabeçalho, sua função é determinar com será o processo de codificação do texto

(encoding), por exemplo:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

O Doctype (Document Type Definition) é a primeira coisa que se deve escrever em um arquivo XHTML, ele vai na primeira linha do seu documento, se deseja-se ter um XML válido, não devemos esquecê-lo, ele serve para informar ao browser que tipo de documento será visualizado. O DOCTYPE informa a DTD que deve ser utilizada para validar o documento apresentado e integra uma linha do cabeçalho. A seguir temos um exemplo desta linha:

```
<!DOCTYPE html PUBLIC "-//w3c/DTD XHTML 1.0 Transitional //
EN" "http://www.w3.org/TR/xhtml1-transational.dtd">
```

O Doctype (Document Type Definition, vulgo DTD) é a primeira linha de um arquivo XHTML, para que o documento possa ser validado, uma vez que ela serve para informar ao browser o tipo de documento a ser visualizado. Existem 3 tipos básicos. O primeiro deles é o Sctrict que é usado quando se deseja um código 100% XHTML e sem erros:

```
<!DOCTYPE htmlPUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Já o transitional é o modo mais usado, pois é utilizado na migração do HTML para o XHTML:

```
<!DOCTYPE htmlPUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Por fim o Frameset é usado quando se utiliza FRAMES em um site:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

EXERCÍCIOS

As questões abaixo devem ser respondidas em forma dissertativa e argumentativa com pelo menos uma lauda. Devem também refletir a interpretação da leitura do texto juntamente com pesquisas sobre o tema arguido.

1. Defina com suas palavras a arquitetura cliente-servidor.
2. Quais as diferenças/semelhanças da arquitetura cliente-servidor com sistemas baseados na web.
3. O que é e para que serve um Servidor de nome de Domínio (DNS – *Domain Name System*)?
4. O que é um cliente web?
5. Defina com suas palavras as principais características da linguagem HTML.
6. O que são e para que serve um marcador em HTML?
7. O que é o CSS (Cascading Style Sheetes – Folhas de Estilo em Cascata)? Explique as diferenças com relação ao HTML.
8. O que é XML?
9. O que é o XHTML?
10. (Questão Prática) Crie uma página HTML que contenha um formulário de cadastro de clientes de uma loja e formate esta página usando CSS.

WEBLIOGRAFIA

Web based x Cliente-Servidor: Porque isso é importante?

<http://www.kitemes.com.br/2012/06/06/web-based-x-cliente-servidor-porque-isso-e-importante-para-minha-empresa/>

Codificação de caracteres no HTML

<http://www.codex.wiki.br/Html/Codifica%C3%A7%C3%A3oDeCaracteres>

Meu primeiro formulário HTML

https://developer.mozilla.org/pt-BR/docs/Web/Guide/HTML/Forms/Meu_primeiro_formulario_HTML

Apostila de XHTML Curso de Internet Professor Rinaldo Demétrio

<http://docplayer.com.br/11698981-Apostila-de-xhtml-curso-de-internet-professor-rinaldo-demetrio.html>

CSS

<http://www.w3c.br/pub/Cursos/CursoCSS3/css-web.pdf>

Desenvolvimento Web com HTML, CSS e JavaScript

http://www.netsoft.inf.br/aulas/5_SIN_Programacao_Web/caelum-html-css-javascript-php.pdf

Apostila de HTML e CSS

<http://www.pereiraaps.com.br/Apostilas/apostilaHTML%20e%20CSS.pdf>

UNIDADE II

DESENVOLVIMENTO DE APLICAÇÕES PARA WEB

Resumo

Desde o advento da Internet, os programadores se apoiaram na linguagem Java como umas das principais tecnologias para desenvolvimento de aplicações na web, principalmente por sua versatilidade de ser executada em diversas plataformas. Passados mais de 20 anos, o Java ainda continua como uma das principais alternativas para os programadores. Contudo, apesar da sintaxe ser muito parecida, o modo como ela é aplicada tornou-se diferente. Tecnologias como o Java Server Pages (JSP), Servlets e Containers emergiram como fortes alternativas frente a crescente ascensão do PHP.

Nesta Unidade veremos os conceitos básicos da linguagem Java, passando pela criação de páginas dinâmicas através do JSP e ainda os Servlets e Containers, além de estudarmos o MVC (Model-View-Controller). O texto desta Unidade contempla ideias de vários autores, expressando diversas visões sobre estes assuntos.

A Unidade é acompanhada de exercícios sem a solução. Cada questão deve ser encarada como um tema, o qual o aluno deve dissertar. Recomenda-se que seja feita uma pesquisa sobre o assunto e que a questão seja respondida de forma ampla, podendo refletir a opinião do aluno. A bibliografia e a webliografia ao fim das unidades devem utilizadas para adquirir um conhecimento razoável sobre o tema de cada capítulo. Ao término da leitura desta Unidade, o estudante deverá: a) Compreender fundamentos da linguagem Java; b) Entender como funciona o JSP, e c) Ser capaz de entender a importância das dos Servlets e Containers.

5 MVC –MODEL VIEW CONTROLLER

O conceito do MVC é extremamente simples, mas a sua visualização não é tão trivial assim. O padrão MVC (*Model-View-Controller*) separa os dados (*Model*) do layout (*View*). Dessa forma, alterações feitas no layout não afetam a manipulação de dados, que por sua vez poderão ser reorganizados sem alterar o layout. O problema é resolvido introduzindo-se um componente entre a manipulação dos dados e a apresentação: o fluxo da aplicação (Controller).

O MVC é usado em padrões de projeto de software, mas abrange mais a arquitetura de uma aplicação do que é típico para um padrão de projeto. Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo, é fácil manter, testar e atualizar sistemas múltiplos (GLAZAR, 2011). Como a parte visual é separada do modelo de negócio, é possível alterar a parte visual sem alterar o sistema todo. É muito simples adicionar novas funcionalidades apenas incluindo seus visualizadores e controles sem alterar o que já foi feito, tornando as aplicações escaláveis. É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são independentes, ganhando em produtividade. Dentre as principais vantagens podemos destacar:

- reaproveitamento de código;
- facilidade de manutenção;
- integração de equipes e/ou divisão de tarefas;
- camada de persistência independente;
- implementação de segurança;
- facilidade na alteração da interface da aplicação;
- aplicação escalável.

Primeiramente vamos entender o papel do M (Model). O Modelo representa os dados da aplicação e as regras de negócio que governam o

acesso e a modificação dos dados. O modelo fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo. Pode ainda ser subdividido em “regras do negócio” e “persistência dos dados”.

Quando pensamos em regras de negócio, estamos pensando no Modelo da aplicação. No Model podemos ter validações, acesso a banco, acesso a arquivos, cálculos, etc. O usuário, por exemplo, coloca um produto em um carrinho de compras e é no Model que faremos o cálculo final do pedido (descontos, juros), que validaremos a conta do usuário, que calcularemos o frete, que validaremos o endereço, etc. As atividades de inserção de um produto em um carrinho de compras e a inserção de endereço para a entrega, por exemplo, são realizadas na View (Visão).

A visão renderiza o conteúdo de uma parte particular do modelo e encaminha para o controlador as ações do usuário; acessa também os dados do modelo via controlador e define como esses dados devem ser apresentados. Apresenta os dados para o usuário sem se preocupar com a origem deles. A View só existe por um único motivo que é mostrar dados (GLAZAR, 2011).

Finalmente, vamos entender o papel do C (Controller). O controlador define o comportamento da aplicação. É ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário. Há, normalmente, um controlador para cada conjunto de funcionalidades relacionadas. O Controller simplesmente delega para o Model as solicitações da View. O Controller não entende as regras de negócio da aplicação. Ele é responsável por saber quem está pedindo algo e a quem enviará este algo.

O Controller conhece a View e conhece o Model, mas o Model não conhece a View, porém a View observa o Model e este avisa quando seus dados foram atualizados, para a View (Figura 8).

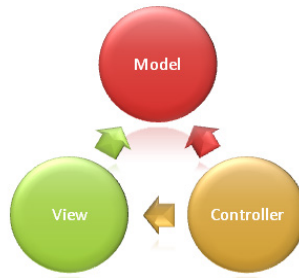


Figura 8: Modelo MVC

O Controller conhece a View e conhece o Model. Isto porque ele recebe as requisições do usuário da View e envia para o Model fazer algo com estas requisições. Exemplo: O usuário quer saber o endereço de uma residência a partir de um cep digitado.

O Model simplesmente recebe a requisição, faz toda a mágica (persiste dados, valida informações, etc) e fica com os dados prontos (atualizados) para serem visualizados novamente pela View. Ela fica observando o Model e quando este é atualizado, a View mostra os dados atualizados para o usuário. E o Model acaba não conhecendo nenhuma das outras camadas. Existem diversos motivos para se trabalhar nesse modelo (GLAZAR, 2011).

Primeiramente, porque, além de elegante, é profissional. A segurança é outro motivo. Temos aqui uma separação bem clara entre a Visualização e as Regras de Negócio. Caso se queira modificar uma regra, como um cálculo de juros, precisa-se apenas modificar o seu Model e não precisaria se preocupar com a visualização. O mesmo vale para a View. Caso se queira modificar uma tela inteira, não precisaria se preocupar com as regras e sim somente com a tela. Uma outra razão muito forte é que pode-se querer disponibilizar os seus métodos em um `WebService`⁸ por exemplo e com uma separação clara do seu modelo, isso seria facilitado.

⁸ É uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis.

Imagine que se tenha uma aplicação Web e outra Desktop, mas que façam a mesma coisa. Seria muito ruim ter funcionalidades iguais nos dois ambientes, pois teríamos que manter as duas iguais. Uma forma seria disponibilizar as funcionalidades em um ponto único e acessá-las nas duas aplicações. Com uma separação clara do modelo poderíamos disponibilizar estas funcionalidades por um Webservice.

O MVC é um padrão de arquitetura que existe, grosso modo, para facilitar a manutenção da nossa aplicação, facilitar a adição de funcionalidades e facilitar a testabilidade da aplicação. Se o desenvolvedor escreve regras de negócio nas suas páginas, deve-se desenvolver com este padrão. Isso trará diferenças na manutenção e na adição de funcionalidades.

6 LINGUAGEM JAVA

Java é uma linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems. Modelada depois de C++, a linguagem Java foi projetada para ser pequena, simples e portátil a todas as plataformas e sistemas operacionais, tanto o código fonte como os binários. Esta portabilidade é obtida pelo fato da linguagem ser interpretada, ou seja, o compilador gera um código independente de máquina chamado byte-code. No momento da execução este byte-code é interpretado por uma máquina virtual instalado na máquina. Para portar Java para uma arquitetura de hardware específica, basta instalar a máquina virtual (interpretador).

Em 1991, um grupo de engenheiros da Sun Microsystems foi encarregado de criar uma nova linguagem que pudesse ser utilizada em pequenos equipamentos como controles de TV, telefones, fornos, geladeiras, etc. Essa linguagem deveria dar a esses aparelhos a capacidade de se comunicar entre si, para que a casa se comportasse como uma federação. Deveria ainda ser capaz de gerar códigos muito pequenos, que pudessem ser executados em vários aparelhos diferentes e praticamente infalíveis (MENGUE, 2008).

Os engenheiros escolheram o C++ como ponto de partida. Linguagem orientada a objetos e gerando pequenos programas, parecia a escolha correta. Para solucionar o problema da execução em várias arquiteturas, eles utilizaram o conceito da máquina virtual mencionado anteriormente, onde cada fabricante iria suportar algumas funções básicas que os programas utilizariam. Até hoje a linguagem resultante deste projeto não é utilizada em aparelhos eletrodomésticos. Ao invés disso, o Java se tornou uma das linguagens de programação mais utilizadas no planeta.

Na maioria das linguagens de programação, é preciso compilar ou interpretar um programa para que ele seja executado em seu computador. A linguagem Java é diferente, pois seus programas são compilados e interpretados. Com o compilador, deve-se inicialmente transformar o programa em uma linguagem intermediária, chamada *bytecode*. Esse código é independente de plataforma, e é mais tarde interpretado por um interpretador Java (MENGUE, 2008). A compilação acontece apenas uma vez; a interpretação acontece todas as vezes que seu programa é executado (Figura 9).

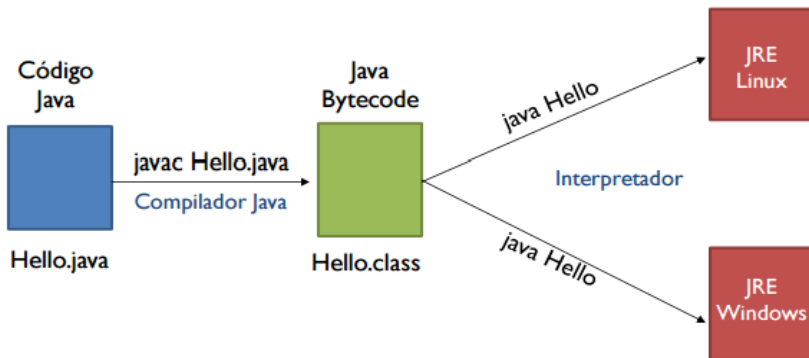


Figura 9: Execução de um programa Java⁹

9 <http://tieconcursos.blogspot.com.br/2015/03/programacao-modular-introducao-java.html>

Deve-se pensar nos *bytecodes* como instruções de máquina para a Java Virtual Machine (ou JVM). Todos os produtos que conseguem executar programas em Java (como um browser que executa applet's) possuem uma cópia da JVM.

Bytecodes Java tornam possível a tecnologia “escreva uma vez, execute em qualquer lugar”. Você pode compilar seu programa Java em qualquer plataforma que possua um compilador. Os bytecodes gerados podem ser interpretados em qualquer plataforma que possua uma JVM (MENGUE, 2008).

6.1 A PLATAFORMA JAVA

Por plataforma, entendemos o conjunto de hardware e software no qual um programa executa. Alguns exemplos de plataformas muito usadas são o Windows, o Linux, o MacOS. A plataforma Java é diferente, pois não envolve hardware; ela utiliza a plataforma de hardware das outras.

A plataforma Java tem dois componentes:

- Java Virtual Machine (Java VM ou JVM);
- Java Application Programming Interface (Java API).

A API Java é uma coleção de componentes de software prontos, que incluem desde estruturas para manipulação de arquivos até a construção de aplicativos gráficos. A API é organizada como um grupo de bibliotecas com classes e interfaces; essas bibliotecas são chamadas de pacotes (MENGUE, 2008).

Para o desenvolvimento de aplicativos utilizando o Java, é necessário a instalação do compilador Java, das API's e da JVM. A instalação do ambiente segue o mesmo esquema da instalação de qualquer produto

para Windows. Devemos fazer o *download*¹⁰ da versão mais apropriada via ftp ou http e executar o arquivo para que o produto se instale.

6.2 MEU PRIMEIRO PROGRAMA JAVA

Como a maioria das linguagens de programação, o código fonte de seu programa em Java deve ser criado a partir de um editor de texto que gere arquivos em formato ASCII. É possível utilizar editores como o Word e o Wordpad, mas o texto deve ser salvo sem formatação. Neste caso, o editor ideal é o notepad.

O programa fonte em Java deve ser salvo obrigatoriamente com a extensão .java . Vamos então criar uma pasta chamada TESTE para que possamos organizar os programas e exercícios que faremos. Execute o editor de texto agora, e vamos digitar nosso primeiro programa em Java. Copie as linhas abaixo:

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

Salve o arquivo como AloMundo.java (letras maiúsculas e minúsculas fazem diferença). A seguir, vamos compilar o programa. A compilação irá gerar os bytecodes. Do prompt do DOS, execute:

```
javac AloMundo.java
```

Se não houver erro, depois de alguns segundos você deve ter acesso ao prompt novamente. Para executar o programa,

```
java AloMundo
```


Percebe-se como resposta a frase “Alo Mundo!”. Isso significa que tudo está certo com seu ambiente (MENGUE, 2008).

6.3 PROGRAMAÇÃO BÁSICA EM JAVA

Nessa seção falamos sobre os conceitos básicos do Java, como tipos de dados, loops, e métodos. Não falaremos sobre a interface gráfica, uma vez que no desenvolvimento web ela é geralmente desenvolvida através de linguagem HTML/CSS. Iremos inicialmente aprender como criar aplicativos ou programas que não utilizam outros recursos além do *prompt* de comando.

6.3.1 Comentários

É sempre interessante a colocação de comentários em programas. Os comentários permitem que a manutenção posterior do código seja mais rápida e serve para indicar o que o programa faz. Os comentários em Java podem ser de dois tipos: usando duas barras (//) e os sinais de /* e */. Utiliza-se duas barras (//) em qualquer posição da linha. Tudo o que aparecer à direita das duas barras será ignorado pelo compilador. Exemplo:

```
a++; // Incremento da variável
```

Existem ocasiões as quais várias linhas de comentário são necessárias. Nesse caso, utilizamos os sinais de /* e */ para indicar início e fim de bloco de comentários, como no exemplo:

```
/* Programa teste  
Tudo que estiver entre aqui é comentário  
*/
```

6.3.2 Tipos de Dados

O Java é uma linguagem a qual necessita que todas as variáveis tenham um tipo declarado. Existem 8 tipos primitivos em Java, destes

seis deles são numéricos, um é o caracter e o outro é o booleano. Os tipos inteiros guardam valores numéricos sem parte fracionária. Valores negativos são permitidos. São eles Int (4 bytes), Short (2 bytes), Byte (1 byte), Long (8 bytes).

Na maioria das ocasiões, o tipo int é suficiente. Não se pode esquecer que como Java é portátil, esses valores são os mesmos para qualquer plataforma de hardware (MENGUE, 2008). Os tipos primitivos a seguir representam valores com ponto flutuante: Float (4 bytes), Double (8 bytes). Normalmente utilizamos o double na maioria das situações as quais são necessárias a representação desse tipo de número, pois sua precisão é maior.

O tipo caracter serve para representar apenas uma letra ou número, bem como para representar caracteres usando a tabela Unicode. Dessa tabela fazem parte a tabela ASCII e mais alguns caracteres especiais. O tipo caracter é sempre representado por aspas simples ('h'). Caracteres representados por aspas duplas ("h") na verdade são strings. O tipo booleano pode assumir apenas dois valores, true ou false. Esse tipo é usado apenas para testes lógicos.

6.3.3 Declaração de Variáveis

A declaração de variáveis em Java, como em várias outras linguagens, exige que o tipo da variável seja declarado. Inicia-se a declaração indicando o tipo da variável e o nome desejado, como no exemplo:

```
int a;  
char ch;
```

Note que todas as declarações terminam com o ponto-e-vírgula que é o sinal usado para indicar o final de comando. Os nomes das variáveis devem ser iniciados com qualquer letra, seguidas por uma sequência de letras ou dígitos. O tamanho do nome da variável não tem limites. É

possível declarar várias variáveis em uma linha, bem como atribuir valores a elas na declaração, como nos exemplos abaixo:

```
int a,b;  
int a = 10; // Isto é uma inicialização
```

6.3.4 Conversões Entre Tipos

Java não tem problemas para atribuir um tipo `int` para um `double` – ele vai tratar o valor como `double`. Assim sempre que uma atribuição for efetuada o tipo mais representativo será utilizado.

Entretanto, existem ocasiões as quais queremos representar o valor inteiro de um tipo `double`, por exemplo. Assim, torna-se necessário converter o tipo em uma operação chamada de `cast`. Essa conversão nada mais é do que indicar o tipo desejado, como no exemplo:

```
double x = 9.345  
int z = (int)x;
```

A variável `z` terá como valor o número 9.

As conversões permitidas sem `cast` são: `byte->short->int->long->float->double` e `char-> int`

6.3.5 Constantes

Pode-se definir constantes em Java utilizando a palavra reservada `final`. Essa palavra indica que se definiu o valor da uma variável e que esse valor não pode ser modificado. Normalmente as constantes são definidas em caixa alta, como no exemplo:

```
final double TEMPERATURA = 25.4;
```

6.3.6 Operadores

Os operadores aritméticos + - * / são utilizados para a adição, subtração, multiplicação e divisão. A divisão retorna resultado inteiro se os operadores forem inteiros, e valores de ponto flutuante em caso contrário (MENGUE, 2008). Se for necessário ter o valor do resto da divisão, utilizamos o % (função mod). É possível utilizar operadores na atribuição das variáveis, como no exemplo:

```
int n = 5;
inta=2*n; //a=10
```

Existe também a possibilidade de utilizar atalhos para operações:

```
x+=4; //equivalenteax=x+4;
```

A exponenciação é feita por uma função da biblioteca matemática. Essa biblioteca tem dezenas de operações específicas.

```
b doubley=Math.pow(x,b); //xéelevadoab(x)
```

6.3.7 Incremento e Decremento

O uso de contadores em programas é muito comum. Existem maneiras de realizar incrementos e decrementos em variáveis utilizando o sinal ++ e o --. Veja nos exemplos:

```
int a = 12;
a++ // a agora vale 13
```

O uso do incremento e do decremento depende da posição onde eles se encontram na expressão. Existem ocasiões em que se quer a expressão calculada e o valor seja incrementado depois. Em outros casos, o valor deve ser incrementado e a expressão avaliada ao final. Acompanhe o exemplo:

```
int m = 7;
int n = 7;
inta=2*++m; //avale16,mvale8
intb=2*n++; //bvale14,nvale8
```

6.3.8 Operadores Relacionais e Booleanos

Esses operadores servem para avaliar expressões. Para verificar a igualdade entre dois valores, usamos o sinal == (dois sinais de igual).

O operador usado para verificar a diferença (não igual) é o !=. Temos ainda os sinais de maior (>), menor (<), maior ou igual (>=), menor ou igual (<=). Existem operadores lógicos AND (&&), OR (||).

6.3.9 Strings

Todos os outros valores que utilizamos em Java com exceção dos tipos explicados acima (ditos primitivos) são objetos (MENGUE, 2008). Um dos objetos mais utilizados é o String (com S maiúsculo). O String é uma sequência de caracteres.

```
String e = ""; // String vazia. Note as aspas duplas.
String oi = "Bom dia";
```

As strings podem ser concatenadas, utilizando o sinal de +, como no exemplo:

```
String um = "Curso";
String dois = "Java";
String result = um + dois;
```

Uma String não deve ser comparada com outra usando o sinal ==, pois elas são objetos. Existe um método especial para comparar objetos, utilizando o equals. O objeto String em Java tem mais de 50 métodos

diferentes. Assim, a comparação da String a com a String b seria:

```
a.equals(b);
```

6.3.10 Vetores e Matrizes

Vetores são estruturas utilizadas para armazenar dados afins. Esses dados podem ser de qualquer tipo, desde variáveis primitivas até objetos complexos. Um vetor pode ser definido assim:

```
int[] vetor = new int[100];
```

Aqui temos um vetor de 100 posições (de 0 a 99) de valores inteiros. Os elementos do vetor podem ser acessados segundo sua posição: `vetor[30]..`

E podemos iniciar seus valores na própria inicialização, como abaixo:

```
int[] impares = {2,3,5,7,9,11,13}
```

É possível definir vetores de várias dimensões. É muito usado em matemática o conceito de matriz, que em Java é definida como:

```
int[][] matriz = new int[5][6];
```

E os valores são acessados da mesma maneira que os vetores.

6.3.11 Controle De Fluxo Do Programa

Apresentamos aqui os comandos que nos permitem controlar o fluxo do programa e expressões condicionais em Java. Mas antes temos que aprender a delimitar blocos e conceituar o escopo. Um bloco nada mais é do que uma série de **linhas** de código situadas entre um abre e fecha chaves `{ }`. Podemos criar blocos dentro de blocos. Dentro de um bloco temos um determinado escopo, que determina a visibilidade e tempo de vida de variáveis e nomes (MENGUE, 2008). Por exemplo:

```

{
    int x = 10;
    // aqui eu tenho acesso ao x
    {
        int z = 20;
        // aqui eu tenho acesso ao x e ao z
    }
    // aqui eu tenho acesso ao x; o z esta
    fora do escopo
}

```

Assim, é permitido a definição de variáveis com mesmo nome, desde que elas não estejam compartilhando o mesmo escopo. A definição dos blocos ajuda a tornar o programa mais legível e a utilizar menos memória, além de indicar quais os comandos a serem executados pelas instruções condicionais e os loop, que veremos a seguir:

```

if (expressão)
    comando ou { bloco }
else // opcional
    comando ou { bloco } // opcional

```

Desvia o fluxo de acordo com o resultado da expressão. Esta pode ser algo simples ou composto. O else é opcional. Se for necessário mais de um comando, é necessário colocar o bloco das instruções entre { } .

O comando return serve para 2 propósitos: mostrar qual valor deve ser retornado do método (se ele não for void) e para encerrar a execução do método imediatamente. Os comandos que utilizamos para executar a mesma porção de código várias vezes são chamados de comandos de iteração ou comandos de loop.

```

while (expressão)
    comando ou { bloco }

```

A expressão é avaliada uma vez antes do comando. Caso seja verdadeira, o comando é executado. Ao final do comando, a expressão é avaliada novamente. Se for necessário mais de um comando, é necessário colocar o bloco das instruções entre `{ }`.

```
do
    comando ou { bloco }
while (expressão);
```

O comando é executado, e a expressão é avaliada no final. A única diferença entre o `do-while` e o `while` é que no primeiro o comando é sempre executado pelo menos uma vez. Se for necessário mais de um comando, é necessário colocar o bloco das instruções entre `{ }`.

```
for (inicialização; expressão; passo)
    comando ou { bloco }
```

Uma variável é iniciada na parte de inicialização. A expressão é testada a cada execução do comando, e enquanto for verdadeira, a(s) instrução(es) contidas no bloco é (são) executada(s). Ao final, passo é executado. É possível a inicialização de mais de uma variável e a execução de mais de uma instrução no passo, dividindo as instruções com vírgulas, como abaixo:

```
for (int i=0, j=1; i < 10 && j != 11; i++, j++)
```

O comando `break` termina a execução de um loop sem executar o resto dos comandos, e força a saída do loop. Já o comando `continue` termina a execução de um loop sem executar o resto dos comandos, e volta para o início do loop para uma nova iteração (MENGUE, 2008).

```
switch (variável)
{
    case (valor1): comando ou { bloco } break; case
(valor2): comando2 ou { bloco2 } break; ...
```



```
default: comando_final ou { bloco final }  
}
```

O comando `switch` serve para simplificar certas situações as quais existem vários valores a serem testados. Assim, identificamos a variável a ser testada, e colocamos uma linha `case` para cada possível valor que a variável pode assumir. No final, permite-se colocar uma linha `default` para o caso da variável não assumir nenhum dos valores previstos. O `break` no final de cada comando serve para evitar comparações inúteis depois de encontrado o valor correto. Se for necessário mais de um comando, é necessário colocar o bloco das instruções entre `{ }`.

6.3.12 Outras instruções

Ainda existem centenas de outras instruções que podem ser utilizadas em Java. A maioria delas faz referência a métodos de objetos e classes, que existem para realizar operações específicas.

Mas não precisamos nos limitar às funções definidas pela linguagem. Temos a possibilidade de criar nossas próprias funções, utilizando procedimentos simples para realizar tarefas complexas (tradicionalmente chamadas de funções). Além disso, podemos ainda estender a API, com classes específicas de outros fabricantes.

6.4 Construção de Programas em Java

Em uma linguagem orientada a objetos como a linguagem Java, fica fácil perceber que é muito mais fácil fazer uma atribuição simples do que criar um método só para alterar o valor de alguma variável. Mas isso tem sentido de ser, em linguagens orientadas a objetos.

A ideia é que o objeto deve gerenciar seus próprios dados, que só devem ser acessíveis ao “mundo exterior” através de seus métodos (exce-

tuando-se aqui os métodos e variáveis estáticas). Então, pelo menos em teoria, cada atributo de um objeto deve ter um método para gravar dados e outro para devolver o dado gravado. Isso vai permitir que esse objeto seja utilizado por qualquer um, a qualquer tempo (MENGUE, 2008).

Vamos passar então um pequeno código de um programa completo em Java para que possamos ir comentando e esclarecendo.

```
import java.util.*;
public class Propriedades {
    public static void main (String[] args) {
        System.out.println ("Bom dia... Hoje é
dia\n");
        System.out.println(new Date());
    }
}
```

Logo na primeira linha temos “import...”. É normal nas primeiras linhas de um programa em Java a colocação da instrução import. Essa instrução serve para que nosso programa possa utilizar qualquer classe externa que ele necessite. No caso, meu programa usa uma função que retorna a data do sistema, e essa função faz parte de java.util. Assim, eu preciso importar essas funções para que meu programa funcione. Isso será explicado melhor quando falarmos a respeito de API's.

Na segunda linha, definimos o nome do objeto (em Java, chamado de classe). É importante notar que o objeto deve ter o mesmo nome do arquivo no disco, caso contrário o programa não irá funcionar. Notamos ainda nessa mesma linha que essa classe é do tipo pública (public). Isso quer dizer que esse objeto é acessível por outros objetos e que ele pode ser importado e usado em outros códigos (MENGUE, 2008).

Na próxima linha, temos a definição do método main. Vemos novamente o public, e a palavra static, indicando que esse método não pode ser instanciado, apenas usado do jeito que está. Vemos que o método main admite uma lista de *Strings* como argumentos (chamado de args).

Esses argumentos não são necessários um programa, mas caso necessitássemos passar algum tipo de informação para qualquer programa, essas informações estariam armazenadas na variável `args`.

Na próxima linha, temos a impressão de um texto na tela. A única novidade é a presença de um sinal `\n` no final do texto. Ele indica ao Java para pular de linha depois de escrever o texto indicado.

Na última linha temos outra impressão. Dessa vez, temos a instanciação de um objeto do tipo `Date`, e seu valor é imediatamente impresso. Como não necessitamos do objeto, apenas queremos um valor impresso, não é necessário criar uma variável apenas para isso. Contudo, nem sempre isso será possível.

6.4.1 Métodos Construtores e Overloading

Como já apresentado, é sempre necessário instanciar um objeto para poder utilizá-lo. Existe um método especial em uma classe que fornece instruções a respeito de como devemos instanciar o objeto, é o construtor. Sua função é garantir que o objeto associado à variável definida será iniciada corretamente. Sempre é necessário tê-lo, e como na maioria das vezes esse construtor não faz nada (além de instanciar o objeto), não há necessidade de declará-lo, pois o Java faz isso automaticamente. O método construtor tem exatamente o mesmo nome da classe. Assim, no exemplo abaixo não o temos definido:

```
class meuObjeto {
    String nome;
    int idade;
    String telefone;
    public void aniversario() {
        idade = idade + 1; }
}
```

Todavia existem casos os quais teremos necessidade de um construtor que faz algo, como na definição de String, que pode ser definida da seguinte forma:

```
String nome = new String();
```

Ou assim:

```
String nome = new String ("Joao");
```

Isso quer dizer que o objeto String tem pelo menos 2 construtores. O primeiro, que inicia o objeto sem argumentos; e outro, que inicia com argumentos (MENGUE, 2008). Esse tipo de artifício é chamado de sobrecarga (em inglês, *Overloading*). Caso queira que o objeto tenha o mesmo tipo de funcionalidade do String, define-se dois construtores da seguinte forma:

```
class meuObjeto {
    String nome;
    int idade;
    String telefone;
    meuObjeto() // Esse é o construtor sem
    argumentos
    {
    }
    meuObjeto(String_nome,int_idade,String_
    telefone)//Construtor com argumentos
    {
        nome = _nome;
        idade = _idade;
        telefone = _telefone;}public void
    aniversario() {
        idade = idade + 1; }
}
```

Agora *meuObjeto* pode ser instanciado como

```
meuObjeto amigo = new meuObjeto();
```

ou

```
meuObjeto amigo = new meuObjeto("Joao", 32,  
"2223311");
```

A sobrecarga é um dos recursos mais interessantes da orientação a objetos (MENGUE, 2008). E não está restrito aos construtores; podemos definir o mesmo nome de método para qualquer método. Assim, tornamos o programa mais legível e temos menos nomes para “inventar”. Como os dois (ou mais) métodos têm o mesmo nome, a diferenciação de qual método é executado depende da quantidade e do tipo dos argumentos enviados. Quando não definimos construtores, o Java cria um sem argumentos para nós. Quando se escolhe definir o construtor, tem-se que definir para todos os tipos, inclusive o sem argumentos.

6.4.2 Utilização das API's

Como já falado, existem muitos objetos prontos que a linguagem Java nos proporciona. Como todos os objetos, eles podem conter dados e vários métodos para que possamos usá-los. É importante para um programador Java conhecer o máximo desses objetos que puder, uma vez que eles facilitam o trabalho na medida em que evitam termos o trabalho de inventar algo que já está pronto. Lembre que normalmente nós não temos acesso aos atributos dessa classe; apenas a seus métodos (MENGUE, 2008).

Quando executamos um `import ...` no início do nosso programa, estamos informando ao compilador Java quais as classes que desejamos usar. Assim, um `import java.util.*`; quer dizer: “eu vou utilizar alguns objetos do pacote `java.util`”. Um programa pode ter tantos `import` quanto necessário. Isso permite que se utilize componentes de pacotes baixados da Internet, com utilização mais restrita.

Conceito De Pacote

Um pacote pode ser entendido como uma série de objetos agrupados por afinidade. Eles ficam “juntos”, pois têm funções semelhantes (como por exemplo manipular texto). Quando criamos objetos para resolver um problema, normalmente estes ficam todos no mesmo subdiretório; entre eles estabelecemos uma relação de “amizade”, e podemos considera-los como parte do mesmo pacote (default). Estando no mesmo subdiretório, certas restrições de acesso entre esses objetos mudam.

6.4.3 Tipos de Métodos: Públicos, Privados e Protegidos

As restrições de acesso em Java existem por dois motivos: O primeiro é não permitir que o programador utilize meus objetos da maneira que quiser. Eu digo o que pode e o que não pode ser utilizado; E o segundo motivo é que tudo que não for público (e conseqüentemente não permite acesso) pode ser alterado na hora que eu quiser. Contanto que os métodos de acesso (que são públicos) fiquem inalterados, ninguém vai perceber a diferença.

Como exemplo, vejamos o objeto String. Este na verdade é um vetor de caracteres. Não há permissão para acessar os conteúdos internos desses caracteres. Isso é feito utilizando um método. Assim, existem situações as quais é interessante que os objetos tenham um certo controle sobre o que o programador pode fazer com eles, para que não ocorram problemas. Para isso, temos três tipos de acesso a métodos e atributos: *public*, *private* e *protected*.

Quando utilizamos a palavra *public*, liberamos o acesso do atributo/método para ser utilizado por qualquer um que importe o pacote ou o objeto. No caso do *private*, ninguém pode acessar o método/atributo daquele objeto, nem mesmo os objetos daquele pacote ou objetos que herdam suas características. Isso evita que certos métodos sejam acessados

diretamente, evitando erros. Métodos/atributos `protected` podem ser herdados, mas não alterados. Exemplo:

```
class exemplo {
    public metodo1() { ... }
    private metodo2() { ... }
}
```

Nessa classe, o `metodo1` pode ser executado por qualquer outro objeto. O método `metodo2` só pode ser executado pelo próprio objeto.

6.4.4 Composição e Herança

Um dos conceitos mais interessantes das linguagens orientadas a objeto é a reutilização de código. Mas para isso realmente funcionar, tem-se que conseguir fazer mais do que simplesmente copiar código e alterá-lo.

Existem duas maneiras diferentes de reutilizar o código. Uma é chamada composição. A composição é geralmente utilizada quando se deseja utilizar as características de um objeto, mas não sua interface. Quando se tem um objeto do tipo `carro` e uma outra pessoa vai utilizar um `carro` com as mesmas funcionalidades, é mais fácil ela construir um `carro` utilizando as “peças” (que já estão prontas). Assim, ela pode importar a classe `carro` e usar:

```
carro meuCarro = new carro();
```

A partir de agora, dentro do seu objeto, tem-se um objeto do tipo `carro`. O se fez aqui foi compor um objeto. Seu objeto agora é do tipo composto, já que ele possui mais do que um objeto dentro dele. Mas existem situações as quais a composição não basta. É quando se deseja utilizar o objeto existente para criar uma versão melhor ou mais especializada dele. Isso é chamado herança (MENGUE, 2008).

A seguir um exemplo de uma classe chamada `Empregado`. Essa

classe tem como atributos o nome, seção e salário do empregado. Existe também um método para alterar o salário.

```
class Empregado
{
    public Empregado (String _nome, String
    _secao, double _salario) {
        nome = _nome;
        secao = _secao;
        salario = _salario;
    } public void aumentaSalario (double
    percentual)
    {
        salario *= 1 + percentual / 100;}
    String nome;
    String secao;
    double salario;}

```

Vamos supor agora que necessitamos de um tipo especial do objeto Empregado, que é o gerente. O Gerente tem secretária, e a cada aumento ele recebe a mais 0,5% a título de gratificação. Mas o Gerente continua sendo um empregado; ele também tem nome, seção e salário. Assim, fica mais fácil utilizar a classe Empregado já pronta como um modelo, e aumentar as funcionalidades.

```
class Gerente extends Empregado
{
    public Gerente (String _nome, String _secao, double _salario, String _secretaria)
    {
        super (_nome, _secao, _salario); //
        Aqui eu chamo a super classe do Gerente
        secretaria = _secretaria;
    }
}

```



```

    }
    public void aumentaSalario (double percentagem)
    {
        super.aumentaSalario (percentagem+0,5);
    }
    public String getSecretaria ()
    {
        return (secretaria);
    }
    public void setSecretaria (String _secretaria)
    {
        secretaria = _secretaria;
    }
    private String secretaria;
}

```

O que fizemos aqui foi literalmente nos aproveitarmos do código de outro programa, especializando o objeto Empregado. Adicionamos métodos a essa classe e mesmo alteramos um de seus métodos (aumentaSalario) para refletir uma nova condição.

Uma palavra que apareceu nessa classe foi a super. Ela referencia a classe da qual essa se originou, a classe que foi estendida ou herdada. Assim, super.aumentaSalario invoca o método aumentaSalario da classe Empregado. A classe Gerente poderia ter substituído completamente o método ou fazer alterações, como fizemos (MENGUE, 2008).

Assim, é sempre uma boa ideia pensar em construir objetos que possam ser genéricos o suficiente para serem reaproveitados. Como referência para nos auxiliar a determinar se devemos utilizar composição ou

herança para construir um objeto, sempre pense:

é para herança (um carro é um veículo).

contém para composição (um carro contém motor, freio, etc.).

Quando não desejamos que um método ou atributo seja redefinido, utilizamos a palavra reservada final (MENGUE, 2008).

6.4.5 Polimorfismo

O conceito de herança nos leva a discutir outro: o polimorfismo. Podemos traduzir essa palavra pela capacidade de um objeto em saber qual o método deve executar. Apesar da chamada ser a mesma, objetos diferentes respondem de maneira diferente.

Assim, quando chamamos `umentaSalario` da classe `Gerente`, é esse método que será executado. Se a classe `Gerente` não tivesse esse método, o método da classe `Empregado` seria executado. Caso a classe `Empregado` também não tivesse esse método, a classe da qual ele veio seria objeto do pedido.

Como exemplo, imagine uma classe chamada `Figura`. Essa classe tem a habilidade de desenhar a si mesma na tela. Se se definir uma classe chamada `Triangulo` que estenda a classe `Figura`, ela pode usar o método da super classe para desenhar a si mesma, sem necessidade de criar um método apenas para isso.

Usando ainda nosso exemplo de `Funcionário` e `Gerente`. Se a classe `Funcionário` tivesse um método para mudar a seção do funcionário, não seria necessário definir um método igual para a classe `Gerente`. Entretanto, quando se invoca o método `Gerente.mudaSecao(nova_secao)`, o objeto saberia como se comportar, pois ele herdou essa “sabedoria” de sua superclasse. Nota: Como todos os objetos definidos são subclasses de `Object`, todas as classes que usamos ou definimos, por mais simples que sejam, têm certas capacidades herdadas desta classe.

6.4.6 Métodos Estáticos

Um último conceito importante diz respeito a métodos especiais. Eles foram criados para resolver situações especiais na orientação a objetos, e também para simplificar certas operações. Imagine uma classe, criado por outra pessoa, que tenha apenas um método. A classe se chama `validaCPF`, e serve para verificar se um CPF é válido ou não.

Segundo as “leis” da orientação a objetos, precisa-se)instanciar um objeto desse tipo e utilizar seu método para a verificação do valor. Isso é um tanto quanto incômodo, pois eu necessita-se simplesmente de uma funcionalidade, e não do objeto todo. Da mesma maneira, temos dezenas de funções matemáticas, físicas, estatísticas e outras que não nos interessam. Gostaríamos apenas de enviar os parâmetros e receber resultados, como se esses métodos fossem funções (MENGUE, 2008).

Nesses casos (e em alguns outros) podemos criar esse método como estático (`static`). Um método estático presente em uma classe não obriga a instanciar um objeto para que se tenha acesso a seus serviços; ele serve apenas para que possamos aproveitá-lo em pequenas computações. Assim, a definição do `validaCPF` seria:

```
static boolean validaCPF(String numero_cpf)
{
    código...
}
```

Quando precisássemos utilizar o código, faríamos algo do tipo: `boolean cpf_valido = validaCPF("123123123");` Os métodos estáticos são utilizados em casos específicos. Um programa orientado a objetos que é feito inteiramente de métodos estáticos não é orientado a objetos.

7 JAVA SERVER PAGES (JSP)

As páginas JSP, ou Java Server Pages, foram criadas para contornar algumas das limitações no desenvolvimento com Servlets: se em um Servlet a formatação da página HTML resultante do processamento de uma requisição se mistura com a lógica da aplicação em si, dificultando a alteração dessa formatação, em uma página JSP essa formatação se encontra separada da programação, podendo ser modificada sem afetar o restante da aplicação (TEMPLE et al, 2004).

Assim, um JSP consiste de uma página HTML com alguns elementos especiais, que conferem o caráter dinâmico da página. Esses elementos podem tanto realizar um processamento por si, como podem recuperar o resultado do processamento realizado em um Servlet, por exemplo, e apresentar esse conteúdo dinâmico junto a página JSP.

Existe também um recurso adicional bastante interessante na utilização de páginas JSP: a recompilação automática, que permite que alterações feitas no código da página sejam automaticamente visíveis em sua apresentação. Assim, não é necessário interromper o funcionamento da aplicação para incorporar uma modificação de layout, por exemplo.

Toda JSP quando compilado se torna uma *Servlet* (capítulo 8). Para que possamos utilizar JSP (Tecnologia Java embutida em páginas) são necessários *Scriptlets*, um pedaço de código em linguagem script (COSTA, 2015). O sufixo *let*, indica o diminutivo, ou seja, pequeno script. Portanto, scriptlet é um pedaço de código Java embutido em um código JSP semelhante a um código HTML.

Através das tags `<% %>` podemos criar variáveis, classes e utilizar praticamente todos os recursos Java. Um Exemplo de código JSP para exibir uma mensagem de boas vindas seria da seguinte forma:

```
<% out.println("Seja Bem Vindo ao Curso de Java para Web"); %>
```

O código acima faria com que o *Scriptlet* escrevesse na página a

mensagem passada para o método `println`. Como toda requisição na Web retorna uma resposta HTML, significa dizer que a mensagem retornada para o usuário do navegador é um HTML, não impedindo que pudéssemos fazer o seguinte:

```
<% out.println("<h1>Seja Bem Vindo ao Curso de Java para Web</h1>"); %>
```

Acima foi colocado as tags do HTML `<h1>`, o que faria com que o texto fosse exibido na página com as características da tag colocada. Dessa forma é possível escrever o texto da forma que se deseja tendo apenas que colocá-lo como saída para o HTML. Para simplificar a escrita o texto ainda poderia ser exibido da seguinte maneira, que também serviria para exibir variáveis e etc.

```
<%= "<h1>Seja Bem Vindo ao Curso de Java para Web</h1>" %>
```

Podemos criar variáveis e exibir os seus resultados como faríamos em uma classe Java comum.

```
<%
    int idade = 51;
    String nome = "João Vieira";
    out.println("Olá, meu nome é "+ nome + "
    e tenho "+ idade + " anos!");
%>
```

7.1 CRIANDO CLASSES

Para criação de classes a regra é a mesma. O código Java deve estar entre as tags JSP. Então para criarmos uma classe que exibisse uma mensagem de boas vindas, teríamos a seguinte classe:

```
<%
    class PrimeiroExemplo { public String
    getMensagem() {
```

```

        return "Olá, Bem Vindo ao Curso de desen-
        volvimento para Web com Java";
    }
} %>

```

Com a classe `PrimeiroExemplo` construída dentro da sua página JSP, podemos instanciar a classe e ver o resultado. Para isso após a criação da classe iremos instanciar a mesma e exibir o resultado como HTML.

```

<%
    PrimeiroExemplo p = new PrimeiroEx-
    emplo();
    out.println(p.getMensagem());
%>

```

7.2 DIRETIVAS

São códigos que demonstram ao container (que pode ser o Tomcat, Jboss ou qualquer outro) informações especiais de processamento concernentes à página. Traduzindo: ela pode ser usada para importar classes, incluir o conteúdo de outra página, especificar a utilização bibliotecas de tags personalizadas, para dizer qual a linguagem de geração de scripts, entre outras coisas (COSTA, 2015).

Como vimos anteriormente, criamos classes dentro do próprio JSP, porém, através das diretivas é que podemos criar ou utilizar classes externas aos JSP's. Para exemplificar, criemos um pacote na pasta WEB-INF do seu aplicativo, após isso crie uma classe. O resultado da classe deve ficar parecido com o código abaixo.

```

package br.cd;
public class Exemplo1 {
    public String getMensagem() {
        return " Olá, Bem Vindo ao Curso de de-

```

```

    desenvolvimento para Web comJava ";
    }
}

```

Após criada a classe, podemos instanciá-la em um JSP da seguinte forma:

```

<%
    Exemplo1 p = new Exemplo1();
    out.println(p.getMensagem());
%>

```

Agora teremos que usar a diretiva de importação para que a classe seja reconhecida no JSP, a diretiva será da seguinte forma:

```

<%@page import="br.cd.Exemplo1"%>

```

Assim podemos utilizar qualquer método que esteja na classe Exemplo1.

7.2.1 Diretiva de Página (page)

A diretiva de página possui diversos atributos com vários propósitos, vejamos eles:

Atributo Info

Usado para inserir informações sumarizadas da página, não havendo restrições ao seu tamanho (COSTA, 2015). Exemplo:

```

<%@ page info="Estudo sobre Diretivas JSP, Prof.
Cristiano Neves, 2006" %>

```

Atributo Language

Usado, em geral, para especificar Java como a linguagem de criação de script para a página. Exemplo:<%@ page language="java" %>

Atributo `ContentType`

Este atributo indica qual o tipo MIME (*Multipurpose Internet Mail Extensions*) da resposta está sendo gerada pela JSP. Os tipos mais comuns são: `"text/plain"`, `"text/html"`, `"text/xml"`. A seguir, o exemplo usado como padrão para as JSPs.

```
<%@ page contentType="text/html" %>
```

Atributo `Extends`

Serve para indicar a super classe que será usada pelo container JSP no momento de tradução da página em um Servlet Java. Exemplo:

```
<%@ page extends="com.taglib.jsp.primeirapagina" %>
Atributo Import
```

Com o atributo `import`, diferente do `extends`, é capaz de estender um conjunto de classes Java que poderão ser usadas nas páginas JSPs. Esta forma é mais econômica de se digitar código, sendo mais prático. Exemplo:

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

Atributo `Session`

`Session` é do tipo boolean, indica se a página está participando do gerenciamento de sessão. Por exemplo, se quisermos dizer que uma página é parte de uma sessão, utiliza-se a seguinte sintaxe:

```
<%@ page session="true" %>
```

Atributo `Buffer`

Responsável por controlar a saída bufferizada para uma página JSP. Se for ajustado para `"none"` o conteúdo de uma JSP é passado instantaneamente à resposta HTTP. O tamanho do buffer é descrito em kilobytes. Exemplo:

Atributo AutoFlush

```
<%@ page buffer="12kb" %> ou
<%@ page buffer="none" %>
```

Semelhante ao Buffer, também é responsável por controlar a saída buferizada, mais exatamente o comportamento do container JSP quando já esteja cheio o Buffer de saída. Neste caso é esvaziado automaticamente o Buffer de saída e o conteúdo enviado para o servidor HTTP que transmite para o Browser responsável pela solicitação. Sendo do tipo boolean, sua sintaxe é dada abaixo:

```
<%@ page autoFlush="true" %>
```

Atributo isThreadSafe

Quando uma página JSP é compilada em um Servlet, ela deve ser capaz de atender a múltiplas solicitações. Para isso devemos utilizar o atributo isThreadSafe, caso contrário é necessário defini-lo como "false". Exemplo:

```
<%@ page isThreadSafe="false" %>
```

Atributo errorPage

ErrorPage indica uma página alternativa que será exibida caso aconteça um erro não previsto durante o processamento de uma página JSP no container. Exemplo:

```
<%@ page errorPage="/diretorio/erro.jsp" %>
```

Atributo isErrorPage

Responsável por definir uma página JSP que servirá como a página de erro padrão para um grupo de páginas JSP. Sendo do tipo boolean, sua sintaxe é descrita abaixo:

```
<%@ page isErrorPage="true"%>
```

Diretiva de Inclusão (include)

Coloca o conteúdo de um arquivo em outro. Não há limites para inclusões, podendo ter várias em uma mesma página. A diretiva se substitui pelo conteúdo indicado. Sua sintaxe é:

```
<%@ include file="URLdoArquivo" %> ou
<jsp:directive.include file="URLdoArquivo" />
```

Diretiva de biblioteca (taglib)

Se usarmos esta diretiva em uma página, um conjunto de tags personalizadas estará a sua disposição, que poderá ser usado em um sistema de página à página. Podendo ser declaradas, também, de duas formas:

```
<%@ taglib uri="URLdaBibliotecadeTags" prefix="PrefixodaTag" %> ou
<jsp:directive.taglib uri="URLdaBibliotecadeTags"
prefix="PrefixodaTag" />
```

7.3 TAGS EM JSP

Existe um tipo de elemento dinâmico que pode ser utilizado em uma página JSP como um elemento de referência a uma biblioteca de Tags (TEMPLE, 2004). Uma biblioteca de Tags permite que você também separe a lógica de programação de sua aplicação do conteúdo da página JSP.

Neste caso, podemos criar um trecho de código e associá-lo a uma tag personalizada. Existem algumas tags que fazem parte do núcleo da linguagem e são chamadas de JSTL (*Java Standar Tag Libs*). Estas permitem o acesso às principais estruturas da linguagem JAVA, como por exemplo, a declaração de variáveis, estruturas de repetição e condicionais, etc. Podemos criar as nossas próprias tags para simplificar a apresentação e permitir que um profissional web (que não programa em java) possa utilizar o nosso código tranquilamente.

7.3.1 Customização de taglibs

O primeiro ponto para o uso de taglibs é definir o que é importante para o sistema que deve ser apresentado em uma página JSP. Se o item aparece com certa frequência, precisa de interação lógica e pode ter seu código simplificado em pequenos blocos. Vamos fazer primeiro alguns exemplos de apresentação de textos simples, para vermos a sintaxe da criação da tag e em seguida faremos um exemplo um pouco mais útil.

Faremos então os seguintes exemplos: Tags sem conteúdo interno, Tags com conteúdo interno, Tags com atributos e Tags sem conteúdo interno. Neste primeiro exemplo, faremos um Hello World simples, ou seja, uma taglib que apresenta sempre este mesmo texto onde for inserida. Vamos criar uma classe chamada `HelloTag.java` que estende a classe pai `SimpleTagSupport`. Essa classe possui os métodos principais para a conexão entre o código e o JSP.

Como método principal, temos o `doTag()`. Nele deve ser inserido o código desejado. Por exemplo, começamos criando um `JspWriter`, para escrever na página JSP e chamamos o método `print()` que receberá como parâmetro a nossa frase Hello World!. Observe que podemos colocar trechos de código HTML na String argumento.

```
package com.br.cacique.taglibs;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class HelloTag extends SimpleTagSupport{

@Override
public void doTag() throws JspException, IOException {
```

```

JspWriter out = getJspContext().getOut();
out.print("<h1>Hello World!</h1>");

}

}

```

Agora vamos criar um arquivo descritor para esta tag, que será responsável por determinar o prefixo utilizado, os atributos (quando existirem), o versionamento, entre outros. Veja o código a seguir:

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.
sun.com/xml/ns/javaee" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance" xsi:schemaLoca-
tion="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.
xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>hello</short-name>
  <uri>/WEB-INF/hello</uri>
  <tag>
    <name>Hello</name>
    <tag-class>com.br.cacique.taglibs.HelloTag</
tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>

```

Perceba que, além das tags para versionamento, criamos uma tag com o nome deste descritor: `<short-name>`; em seguida, indicamos a uri (identificador do recurso), que é o caminho para este arquivo; e,

por fim, declaramos a tag (TEMPLE, 2004). Para isso, definimos o seu nome, a classe relacionada e tipo de conteúdo `<body-content>`, que nesse caso recebe o valor `empty`, pois não temos conteúdo associado à tag.

Por fim, a utilização. O primeiro passo é importar a sua biblioteca. Veja a 2ª linha do código a seguir. Estamos atribuindo a descrição que criamos ao prefixo `h` (você pode escolher o prefixo que melhor se adapte ao seu código).

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="h" uri="/WEB-INF/hello.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h:Hello/>
</body>
</html>
```

Para utilizar a biblioteca, basta então chamarmos a tag no local desejado. Por exemplo, a colocamos diretamente no corpo da página. Como esta tag não tem nem corpo, nem atributos, basta chamá-la pelo seu nome, com fechamento na mesma tag. Assim:

```
<h:Hello/>
```

Neste caso, observe que usamos o prefixo `h`, seguido do nome da tag que definimos no descritor. Ao executar o projeto, deve-se ser capaz de enxergar a frase `Hello World!` como um título.

7.3.2 Tags com conteúdo interno

Vamos dizer agora que a nossa taglib vai receber o valor de seu conteúdo e apresentá-lo como um título. Novamente, este código é bem simples e nem precisaria de uma taglib para tal, mas estamos fazendo isso para estudar o seu funcionamento. Para que a tag possa pegar o conteúdo interno das tags de abertura e fechamento, precisamos fazer algumas poucas alterações. Primeiro, vamos alterar o descritor, dizendo que o body-content agora possui algo. Observe a alteração na destacada no código a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
  <taglib version="2.1" xmlns="http://java.
sun.com/xml/ns/javaee" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance" xsi:schemaLoca-
tion="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.
xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>hello</short-name>
  <uri>/WEB-INF/hello</uri>
  <tag>
  <name>Hello</name>
  <tag-class>com.br.cacique.taglibs.HelloTag</
tag-class>
  <body-content>scriptless</body-content>
  </tag>
</taglib>
```

Observe que no arquivo `HelloTag.java`, criamos agora um `StringWriter` e atribuímos a ele o conteúdo da nossa tag através dos métodos `getJspBody()` (que pega o conteúdo da tag) e `invoke(sw)`

(que coloca este conteúdo no `StringWriter sw`).

```
package com.br.cacique.taglibs;
import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class HelloTag extends SimpleTagSupport{
    private StringWriter sw = new StringWriter();
    @Override
    public void doTag() throws JspException, IOExcep-
    tion {
        getJspBody().invoke(sw);
        JspWriter out = getJspContext().getOut();
        out.print("<h1>" + sw.toString() + "</h1>");
    }
}
```

Observe que a utilização da tag no index será diferente, pois ela terá um conteúdo que é o texto a ser mostrado.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="h" uri="/WEB-INF/hello.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/
html; charset=UTF-8">
<title>JSP Page</title>
</head>
```

```

<body>
<h:Hello>Olá Mundo!</h:Hello>
</body>
</html>

```

7.3.3 Tags com atributos

Agora vamos adicionar um atributo a esta tag que criamos. Digamos que queremos passar a cor do texto apresentado (TEMPLE, 2004). Assim, teremos uma tag com conteúdo interno, que será o texto, e um atributo que conterà a cor. Novamente, precisamos alterar o descritor da taglib para que tenha a regra do atributo.

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib      version="2.1"      xmlns="http://java.
sun.com/xml/ns/javaee"      xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"      xsi:schemaLoca-
tion="http://java.sun.com/xml/ns/javaee http://java.
sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd">
  <tlib-version>1.0</tlib-version>
  <short-name>hello</short-name>
  <uri>/WEB-INF/hello</uri>
  <tag>
  <name>Hello</name>
  <tag-class>com.br.cacique.taglibs.HelloTag</ta-
g-class>
  <body-content>scriptless</body-content>
  <attribute>
  <name>color</name>
  <required>true</required>
  </attribute>
  </tag>
</taglib>

```


Para o element <tag> adicionamos um novo elemento <attribute> que contém alguns possíveis parâmetros, a saber:

- name - nome do atributo
- required - se é um atributo obrigatório (true | false)
- type - tipo de dados do atributo (por padrão é String)
- description - informação do atributo
- fragment - declara se o atributo deve ser tratado como um JspFragment

Para o nosso exemplo, criou-se um atributo chamado color que é obrigatório. Observe agora como tratamos o HelloTag.java:

```
package com.br.cacique.taglibs;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

/**
 *
 * @author cacique
 */
public class HelloTag extends SimpleTagSupport{

    private String color = "#000";
    public void setColor(String color) {
this.color = color;
    }
}
```

```
private StringWriter sw = new StringWriter();

@Override
public void doTag() throws JspException, IOException {
    getJspBody().invoke(sw);
    JspWriter out = getJspContext().getOut();
    out.print("<h1 style='color:" + color + "'>" + sw.
toString() + "</h1>");
}

}
```

Para que o atributo seja acessado no método principal, é necessário que haja uma variável global com o mesmo nome, e um método Setter para ela. Caso a variável não seja obrigatória, lembre-se de tratar o caso em que ela seja nula.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="h" uri="/WEB-INF/hello.tld" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h:Hello color="#f00">Olá Mundo</h:Hello>
</body>
</html>
```

Assim como no HTML, um atributo de tag deve ser encarado como um par: nome + valor (entre aspas). Poderíamos então colocar um tratamento para os possíveis valores, a fim de evitar erros. Mas essas são melhorias que podemos fazer além do básico explorado aqui.

8 SERVLETS E CONTAINERS JAVA

Um Servlet container é similar ao chamado *Web Container*. Em Java EE (*Java Enterprise Edition*), o container contém os componentes construídos como Servlets (container para aplicações Web) ou EJBs (Enterprise JavaBeans) – container para componentes de negócio. Um exemplo de container para Web é o Tomcat¹¹. Quando uma aplicação web faz uma solicitação para um Servlet, o servidor não entrega a solicitação diretamente ao Servlet, mas sim ao container que contém o Servlet. O container gerencia o ciclo de vida, dá suporte ao *multithread*¹², segurança e suporte para páginas JSP (Java Server Pages).

Já um Web server ou app server é servidor de aplicação ou em inglês, *application server*, que é um software o qual disponibiliza um ambiente para a instalação e execução de certas aplicações. Os servidores de aplicação também são conhecidos como software de *middleware*. Seu objetivo é disponibilizar uma plataforma, que abstraia do desenvolvedor de software algumas das complexidades de um sistema computacional. No desenvolvimento de aplicações comerciais, por exemplo, o foco dos desenvolvedores deve ser a resolução de problemas relacionados ao negócio da empresa, e não questões de infraestrutura da aplicação. O servidor de aplicações responde a algumas questões comuns a todas as aplicações, como segurança, garantia de disponibilidade, balanceamento de carga e tratamento de exceções.

11 <http://tomcat.apache.org/>

12 Processamento paralelo de processos

8.1 SERVLETS

A tecnologia *Servlet* foi introduzida pela Sun Microsystems em 1996, aprimorando e estendendo a funcionalidade e capacidade de servidores Web. *Servlets* é uma API para construção de componentes do lado servidor com o objetivo de fornecer um padrão para comunicação entre clientes e servidores (COSTA, 2015).

Servlets são classes Java que são instanciadas e associadas com servidores Web, elas são capazes de processar requisições e respostas do protocolo HTTP. Dessarte, uma (classe) *Servlet* necessita de um container Web para ser executado, o container gerencia as instâncias dos *Servlets* e provê os serviços de rede necessários para as requisições e respostas. Tipicamente existe apenas uma instância de cada Servlet, no entanto, o container pode criar várias threads de modo a permitir que uma única instância Servlet atenda mais de uma requisição simultaneamente. Assim de modo simples, podemos entender que uma servlet funciona como um pequeno servidor que recebe chamadas de diversos clientes, essa servlet é um objeto Java instanciada pelo servidor.

Servlets recebem requisições (request) e retornam respostas (response), respostas essas que podem ser qualquer conteúdo desde uma página HTML até arquivos de conteúdo em formato XML. Na verdade os *Servlets* podem trabalhar com vários tipos de servidores e não só servidores Web, uma vez que a API dos Servlets não assume nada a respeito do ambiente do servidor, sendo independentes de protocolos e plataformas (COSTA, 2015).

Para que uma classe Java seja denominada Servlet é necessário que a mesma esteja ligada à interface `javax.servlet.Servlet`, mais especificamente à classe `javax.servlet.http.HttpServlet` responsável pelo uso de servlets com o protocolo HTTP.

8.2 CICLO DE VIDA E ARQUITETURA DAS SERVLETS

A tecnologia *Servlet* foi projetada para contornar problemas do antigo CGI, *Servlets* são carregadas na memória uma única vez quando o servidor é inicializado ou quando a *Servlet* é requisitada. Como o *servlet* é instanciado uma única vez, o container é responsável por criar threads para atender requisições simultâneas que cheguem as *Servlets*.

Uma *servlet* segue um ciclo de vida composto de 3 fases: inicialização, atendimento de requisições e finalização. A inicialização ocorre quando o container é iniciado, ou quando a *Servlet* recebe a primeira requisição. A segunda fase é o atendimento de requisições que ocorre quando qualquer requisição é feita a *servlet* e a fase de finalização ocorre quando o servidor é parado, liberando da memória a instância da *servlet* (COSTA, 2015).

Para construir uma *servlet* é necessário que a classe estenda `javax.servlet.http.HttpServlet` onde a classe será capaz de receber requisições dos métodos HTTP como GET e POST. Dessa forma a classe *servlet* se torna capaz de processar requisições sendo possível reescrever métodos java referentes aos tipos de requisições GET e POST do HTTP. No código a seguir, existe ainda um terceiro método da API *servlet* capaz de processar ambas requisições HTTP.

```
package br.ac.actions;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class PrimeiraServlet extends HttpServlet {
    @Override
    protected void doGet (HttpServletRequest
        request, HttpServletResponse response)
```

```

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

```

Os métodos reescritos da classe `HttpServletRequest` fornecem dois parâmetros do tipo `HttpServletRequest` e `HttpServletResponse`. `HttpServletRequest` fornece acesso a todas as informações de requisições feitas por clientes. Na tecnologia JSP o conhecíamos por Objeto Implícito `request`, já o `HttpServletResponse` é responsável por produzir respostas que vão desde um conteúdo HTML até um Cookie para máquina cliente. Em JSP o Objeto Implícito `response` o representava.

Para ter acesso a todas as fases do ciclo de vida de uma servlet, utilizamos a reescrita de método de inicialização denominado `init()`, com ele podemos fazer leitura de parâmetros de configuração do `web.xml`, inicialização de variáveis de classe (variáveis estáticas), inicialização de conexões com banco de dados e etc. Veja as assinaturas do método de inicialização em uma classe `Servlet`.

```

@Override
public void init(){
    System.out.println("Servlet Iniciada...");
}

```

É possível ainda outra assinatura, passando um objeto `javax.servlet.ServletConfig`, é através desse parâmetro que o `Servlet` pode obter os parâmetros contidos no `web.xml`.

```
@Override
public void init( javax.servlet.ServletConfig config
){
System.out.println("Servlet Destruída...");
}
```

A finalização de uma *Servlet* é tratada pela implementação do método `destroy()`, esse método é disparado no instante em que a *Servlet* é destruída, seja pelo servidor ou pela própria aplicação. Nesse método podem ser executadas rotinas de finalização, como encerramento de conexões com banco de dados, finalização de threads e etc. A assinatura do método `destroy` é a seguinte.

```
@Override
public void destroy(){
}
```

Além dos métodos citados anteriormente, existem alguns outros que não são muito comuns e necessitam de um contexto específico e conhecimento avançado sobre o protocolo HTTP (COSTA, 2015).

```
public void doDelete(HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response) {
}
public void doPut(HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response) {
}
public void doOptions(HttpServletRequest request,
```

```

    javax.servlet.http.HttpServletResponse response) {
    }
    public void doTrace(HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response) {
    }

```

8.3 SERVLETS E SUAS HIERARQUIAS

Como sabemos, uma Servlet é uma classe java capaz de receber e enviar requisições, para isso a classe `javax.servlet.Servlet` fornece uma hierarquia (Figura 10) de classes capazes de receber requisições de vários protocolos. A classe `javax.servlet.GenericServlet` é capaz de atender requisições genéricas de qualquer protocolo, já a classe `javax.servlet.HttpServlet`, de `GenericServlet` atende requisições em HTTP (COSTA, 2015).

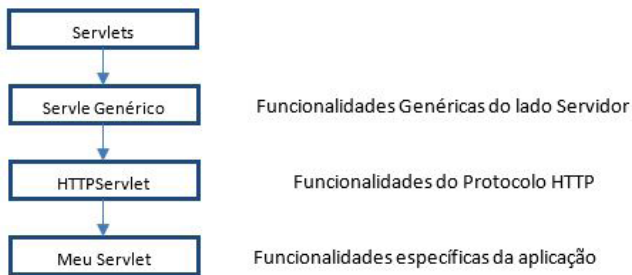


Figura 10: Hierarquia de uma Classe Servlet

8.4 MAPEAMENTO DE SERVLETS

Para que seja possível uma Servlet atender requisições, além do que já foi mostrado, é necessário também fazer o que chamamos de Mapeamento de Servlets. A especificação de Servlet define um arquivo chamado `web.xml` que será o responsável por conter esses mapeamentos, além de

várias outras configurações de uma aplicação web (COSTA, 2015). O arquivo web.xml é um arquivo construído sobre as regras da tecnologia XML.

Para mapear uma servlet é necessário dois elementos: o `<servlet>` e o `<servlet-mapping>`. Veja a seguir quais outros elementos os mesmo podem possuir:

<servlet>

<servlet-name> deve conter o nome da Servlet escolhido pelo desenvolvedor; **<servlet-class>** deve conter o nome da classe (incluindo a informação sobre o pacote, se existir); **<init-param>** deve conter um parâmetro de inicialização do Servlet; pode haver nenhum, somente um, ou mais de um elemento deste tipo para cada Servlet.

<load-on-startup> deve conter um inteiro positivo indicando a ordem em que a Servlet será carregada em relação a outras existentes na aplicação. Inteiros menores são carregados primeiro; se este elemento não existir, ou seu valor não for um inteiro positivo, fica a cargo do Servlet Container decidir quando o Servlet será carregado (possivelmente, no instante em que for feita a primeira requisição a esse Servlet). Dessa forma poderíamos iniciar um mapeamento do elemento `<servlet>` para uma servlet da seguinte forma:

```
<servlet>
    <servlet-name>servlet</servlet-name>
    <servlet-class>br.ac.action.Servlet</
servlet-class>
</servlet>

<servlet-mapping>
```

<servlet-name> Nome que faz a ligação com o elemento `<servlet-name>` da tag `<servlet>`. **<url-pattern>** URL pela qual a Servlet será acessada. É possível ainda definir alguns caracteres especiais como *, com o valor /*. Nesta tag toda requisição feita ao servidor será redirecionada para a servlet mapeada, ou ainda mapear utilizando

o valor `/*.jsp` onde qualquer página `.jsp` requisitada, a Servlet receberá a requisição. O Mapeamento do elemento `<servlet-mapping>` pode ser construído como abaixo.

```
<servlet-mapping>
    <servlet-name>servlet</servlet-name>
    <url-pattern>/PrimeiraServlet</url-pattern>
</servlet-mapping>
```

9 ACESSO AO BANCO DE DADOS

O banco de dados é onde guardamos os dados que pertencem ao nosso sistema. A maioria dos bancos de dados comerciais hoje em dia são relacionais e derivam de uma estrutura diferente daquela orientada a objetos. O MySQL¹³ será o banco de dados que usaremos para nossos exemplos. Para aqueles que não conhecem um banco de dados, é recomendado ler mais sobre o mesmo e SQL para começar a usar a api JDBC.

9.1 UTILIZANDO JSP E JAVABEANS PARA ACESSAR O MYSQL

Javabeans são classes que possuem o construtor sem argumentos e métodos de acesso do tipo `get` e `set`. Segundo a especificação da Sun Microsystems, os JavaBeans são “componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento”. Um bean também pode ser definido como uma classe Java que expõe propriedades, seguindo uma convenção de nomenclatura simples para os métodos `getter` e `setter`. Já os EJBs (Enterprise JavaBeans) são javabeans com características mais avançadas.

13 <https://www.mysql.com/>

Podemos usar beans por diversos motivos, normalmente as classes de modelo da nossa aplicação costumam ser javabeans.

Agora vamos realizar um exemplo de conexão com banco de dados, fazendo com que uma classe faça a conexão e o JSP acesse o resultado através do mesmo.

```
Package conexao.jdbc;
Public class ConnectionFactory {
    Publicstatic Connection getConnection()
    throws SQLException {
        Try{
            Class.forName("jdbc:mysql://localhost/
            teste");
            ReturnDriverManager.getConnection(
            "com.mysql.jdbc.Driver","root","");
        } catch (ClassNotFoundException e) {
            thrownewSQLException(e.getMessage());
        }
    }
}
```

Cada vez que o programador precisasse abrir a conexão bastaria ele dar o comando `Connection con = ConnectionFactory.getConnection();`

Antes de poder fazer consultas ao banco de dados com JDBC, é preciso criar um objeto `Statement` (HARTKE NETO, 2002). Um objeto `Statement` pode ser reutilizado através de múltiplas requisições SQL. É criado pelo objeto `Connection` e é, portanto, dependente do driver que você está usando para conectar ao banco de dados:

```
Statement statement = connection.createStatement();
```

9.1.1 Fazendo consultas com JDBC

Fazer uma consulta com JDBC é fácil uma vez que se tenha o objeto Statement: se passa a consulta como parâmetro do método `executeQuery()`. A tabela de resultados é retornada pelo método como uma instância da classe `ResultSet` da API de JDBC. Abaixo está um exemplo de consulta:

```
String query = ``SELECT * FROM TABELA``;  
ResultSet resultset = statement.executeQuery(query);
```

O objeto `ResultSet` revela o conteúdo da tabela uma linha por vez e fornece métodos para acessar dados de cada coluna da tabela da linha corrente, indicada por um cursor interno. Quando o `ResultSet` é criado, o cursor de linha não está na primeira linha, mas sim antes dela (HARTKE NETO, 2002). Chamando o método `next()`, o `ResultSet` avança o cursor em uma linha, retornando valor verdade se a chamada foi bem sucedida. Por exemplo, para passar por todas linhas do `ResultSet`, faríamos o seguinte:

```
while (resultset.next()) {  
    System.out.println ("Mais uma linha!");  
}
```

Quando o cursor está posicionado em uma linha que você gostaria de examinar, pode-se chamar vários métodos suportados pelo objeto `ResultSet` para restaurar dados das colunas e obter informações sobre os resultados.

Mais adiante, na sessão 10.8, veremos outros comandos para manipulação de informações no banco de dado. Apesar de serem vistos no contexto do PHP, eles podem ser facilmente portados para o JSP usando os comandos vistos acima.

EXERCÍCIOS

As questões abaixo devem ser respondidas em forma dissertativa e argumentativa com pelo menos uma lauda. Devem também refletir a interpretação da leitura do texto juntamente com pesquisas sobre o tema arguido.

1. Explique, de forma detalhada, o conceito de Model View Controller.
2. Explique como se dá a execução de um programa Java, desde seu desenvolvimento até a execução na máquina do cliente.
3. Quais as diferenças entre Métodos Públicos, Privados e Protegidos?
4. O que são os chamados “Pacotes” em Java?
5. O que é um Servlet?
6. O que é o Java Server Pages? Qual sua diferença de uma aplicação Java convencional?
7. Explique as diretivas de página em uma aplicação JSP?
8. O que é o JDBC?
9. (Questão Prática) Escreva uma página em JSP que contenha um formulário com uma série de campos para cadastro de clientes e que submeta esse formulário a gravação em um banco de dados com os mesmos campos.

WEBLIOGRAFIA

MVC simples e prático

<http://www.k19.com.br/artigos/mvc-simples-e-pratico-parte-i/>

Apostila de Java

www.unicamp.br/fea/ortega/info/cursojava/java-pet.doc

Java e Orientação a Objetos – Caelum

<https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf>

Introdução a Java

<http://tieconcursos.blogspot.com.br/2015/03/programacao-modular-introducao-java.html>

Tag Libs

<http://pedrocacique.com/blog/2016/04/taglib.php>

Servlet container, Web container e Web Server

<http://www.guj.com.br/t/servlet-container-web-container-e-web-server/66438>

Java para Desenvolvimento WEB

<http://www.inf.ufrgs.br/~tsrodrigues/utilidades/javaWeb.pdf>

UNIDADE III

PHP e Linguagem de Scripts

Resumo

*Com o advento da WEB 2.0¹⁴, a partir do qual houve um grande apelo da interação e adaptação das páginas web de acordo com a utilização do usuário, foram necessárias a criação de linguagens que permitiam o dinamismo das aplicações na internet. Isto foi possível com o surgimento das linguagens sever-side (lado-servidor). Neste caso, ao acessar as páginas, comandos eram executados, “montando” uma a própria página na hora de acordo com a interação do usuário. Com o passar dos anos, o PHP foi ganhando cada vez mais funcionalidades que ajudariam no desenvolvimento das páginas dinâmicas. Além disso, também ganhou uma série de características que possibilitaram usos adicionais do PHP, não sendo utilizado só no desenvolvimento de sites, mas se tornando **uma** das principais linguagens quando se fala em desenvolvimento web.*

Nesta Unidade veremos os principais conceitos da linguagem PHP. Além disso, apresentaremos também o JavaScript, responsável por principalmente controlar ações do usuário dentro da página web.

O Capítulo é acompanhado de exercícios sem a solução. Cada questão deve ser encarada como um tema, o qual o aluno deve dissertar. Recomenda-se que seja feita uma pesquisa sobre o assunto e que a questão seja respondida de forma ampla, podendo refletir a opinião do aluno. A bibliografia e a webliografia ao fim dos capítulos e unidades devem ser utilizados para adquirir um conhecimento razoável sobre o tema de cada capítulo.

14 termo popularizado a partir de 2004 para designar uma segunda geração de comunidades e serviços, tendo como conceito a "Web como plataforma", envolvendo wikis, redes sociais, blogs e Tecnologia da Informação.

10 PHP

PHP (*Hypertext Preprocessor* ou *Personal Home Page*) é uma linguagem que permite criar sites web dinâmicos, fundamentada nos dados submetidos pelo usuário e derivada dos dados contidos no banco de dados, que são alterados frequentemente. Tomar-se-á como exemplo uma loja virtual (GLAZAR, 2011). Os produtos estão sempre sofrendo alterações, seja no preço, na quantidade em estoque, nos produtos em promoções, nos lançamentos, etc. Hoje, quando se acessa em uma loja virtual, tem-se alguns produtos em promoção, outros em lançamentos, com um determinado preço. Na próxima semana que você visitar o site, pode ser que os preços estejam mais baixos, por causa de novas promoções, ou aquele produto que foi visto não se encontra mais em estoque, já tenha sido vendido.

Passaremos a programar para web sob a visão do servidor. Para isso, utilizaremos a linguagem PHP. Como pré-requisito, é necessário o conhecimento de HTML (capítulo 2), principalmente de formulário, que será utilizado para enviar dados para o servidor.

O código PHP é executado no servidor, sendo enviado para o cliente apenas HTML. Dessa maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente.

O PHP foi criado em 1995 por Rasmus Lerdorf com o nome de Personal Home Page Tools (Ferramentas Para Página Pessoal), para auxiliar no desenvolvimento de páginas simples. Como teve boa aceitação e muitos programadores utilizando-as, novas versões foram desenvolvidas com cada vez mais recursos. Existem outras linguagens de programação que podemos utilizar para criar as páginas dinâmicas, como Java (visto anteriormente), Perl, ASP, etc.

Para testar as páginas PHP, não basta dar um duplo clique nos “arquivos. php”, como se faz com os .htm ou .html. É necessário ter um ser-

vidor web configurado para isso. Um dos servidores web mais utilizados é o Apache. Pode-se instalar o Apache e o PHP separados. Para isso basta pegar os arquivos de instalação nos respectivos sites oficiais.

Porém, configurações manuais deverão ser feitas para os dois funcionarem perfeitamente. A forma mais fácil de instalar é utilizar pacotes que instalam e configuram todos os programas necessários para o desenvolvimento de páginas web de uma única vez (GLAZAR, 2011). Um conjunto muito utilizado consiste do Apache (servidor web), MySQL (banco de dados) e PHP (linguagem para as páginas web dinâmicas), conhecido como AMP (inicial de cada produto). Quando esses produtos são instalados no Linux, chamamos de LAMP¹⁵. Quando são instalados no Windows, chamamos de WAMP¹⁶.

Em qualquer instalação, seja no Windows ou no Linux, um diretório específico será criado para colocar as páginas em PHP, chamado *www*. Quando o Apache recebe uma solicitação para exibir uma página, ele irá buscar nesse diretório.

No Windows, o diretório *www* fica dentro do diretório de instalação do produto. Por exemplo, se você usou o EasyPHP, o diretório é: `C:\Arquivos de programas\EasyPHP-5.3.2\www`

O diretório “EasyPHP” pode mudar de nome de acordo com a versão instalada. No Linux, o diretório é `/var/www`

Podem ser criados subdiretórios dentro do diretório *www*. É até recomendado que se faça isso, para organizar melhor as páginas. Para testar o ambiente, primeiro devemos verificar se os programas estão em execução. Pode-se configurá-los para iniciar automaticamente, quando o computador for ligado, ou manualmente.

Agora vamos criar um arquivo com extensão `.php` (teste.php, por exemplo) na pasta base *www*. Abra-o com qualquer editor de texto e digite: `<?php phpinfo(); ?>`

15 <http://lamphowto.com/>

16 <http://www.wampserver.com>

Salve-o e em seguida digite em seu navegador favorito o seguinte endereço:

`http://localhost/teste.php`.

`teste.php` é o nome do arquivo PHP que foi criado. *localhost* significa que o seu navegador irá procurar o arquivo no seu próprio computador, no diretório `www` configurado na instalação. Na internet, como o servidor está em outro local, substitui-se `localhost` pelo endereço web da empresa, como por exemplo: `http://www.empresaxyz.com.br/cadastro.php`

Se a instalação estiver correta, uma tela com informações sobre a configuração do PHP deverá ser exibida, como indicado pela Figura 11 a seguir.


PHP Version 5.2.4-2ubuntu5.1 	
System	Linux XenacoDTS 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686
Build Date	May 9 2008 16:14:00
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini, /etc/php5/apache2/conf.d/pdo_pgsql.ini, /etc/php5/apache2/conf.d/pgsql.ini, /etc/php5/apache2/conf.d/ldap.ini, /etc/php5/apache2/conf.d/xmlrpc.ini, /etc/php5/apache2/conf.d/xsl.ini
PHP API	20041225
PHP Extension	20060813
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, fts
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

Figura 11: Configurações do Servidor gerada pela função `phpinfo()`.

10.1 SINTAXE DO PHP

Para diferenciar o código PHP dentro da página em HTML, podem ser utilizados os seguintes delimitadores, sendo o primeiro o padrão. O segundo é uma simplificação do primeiro. O terceiro segue o estilo de scripts em HTML. O quarto segue o estilo do ASP (GLAZAR, 2011).

1.

```
<?php
```

Comandos;

```
?>
```

2.

```
<?
```

Comandos;

```
?>
```

3.

```
<script language="php">
```

Comandos;

```
</script>
```

4.

```
<%
```

Commandos;

```
%>
```

Para utilizar a forma simplificada, bem como o estilo ASP, o arquivo de configuração `php.ini` deve ser alterado, pelos campos `short_open_tags` e `asp_tags`, respectivamente.

Como o código PHP é processado no servidor e apenas o HTML é enviado como resposta, utilizaremos o comando `echo` para gerar esse HTML:

```
<html>
  <head>
    <title>Exemplo echo PHP</title>
  </head>
  <body>
    <?php
      echo "<p>Teste do echo</p>";
      echo "<p>Tchau!</p>";

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

Todos os arquivos devem ser salvos com a extensão `.php` no diretório `www` da instalação do servidor web. Crie subdiretórios dentro do `www` para melhor organizar seus arquivos. Os subdiretórios dentro do diretório `www` devem ter permissão de gravação e escrita.

Depois de programar e salvar o arquivo, vamos testá-lo. Para isso, abra seu navegador favorito e digite: `http://localhost/nomedo-arquivo.php`, onde:

`localhost` – corresponde ao seu computador local; e `nomedoarquivo.php` – nome que você deu no seu arquivo PHP (no nosso exemplo, `PrimeiraPagina.php`). Se o servidor web estiver em outro computador, então troque `localhost` pelo IP (ou `hostname`) desse servidor. Caso tenha criado subdiretórios, então acrescente no endereço do navegador os subdiretórios criados. Exemplo: `http://localhost/ProgWeb/`

`cadcliente.php` Nesse exemplo foi criado o subdiretório `ProgWeb` e dentro dele foi colocada a página `cadcliente.php`.

10.2 VARIÁVEIS E TIPOS

As variáveis em PHP não precisam ser declaradas. Quando atribuímos algum valor para elas, o tipo é automaticamente reconhecido. Os tipos suportados são:

- Inteiros (integer ou long);
- Reais (float ou double);
- Strings;
- Array (vetores);
- Objetos

Os nomes das variáveis devem ser criados com um '\$' seguido de uma string que deve ser inicializada por uma letra ou '_' como por exemplo:

```
$x = 10.4;
$frase = "Exemplo de variável string";
$_cont = 0;
```

O PHP é *case sensitive*, ou seja, letras maiúsculas são diferentes de minúsculas. Portanto, para facilitar, deve-se criar variáveis sempre em minúsculo (GLAZAR, 2011). Os comentários podem ser de uma linha, utilizando o símbolo `//`, ou de mais linhas, delimitado pelos símbolos `/*` e `*/`. Exemplos:

```
$cont = 0; // Exemplo de comentário de uma linha
```

```
/* Exemplo de comentário com mais de uma linha.
Preste atenção nos símbolos delimitadores. */
```

10.3 ESTRUTURAS DE CONTROLE

As estruturas de controle servem para controlar a ordem de execução das instruções de um programa. As principais são as de seleção e repetição.

10.3.1 Comandos de seleção

Os comandos de seleção servem para escolher um determinado bloco de comandos a partir da avaliação de uma expressão. Os comandos de seleção são: `if` (e suas variações) e o `switch`. Um bloco de comandos é delimitado pelos símbolos: '{' e '}'.

O `if` – executa um bloco de comando caso a expressão seja verdadeira (se.. então). Já o `if...else` – executa o primeiro bloco de comandos se a expressão for verdadeira, e o bloco do `else` caso a expressão seja falsa. (se...então...senão).

`if...elseif...else` é utilizado quando várias condições precisam ser analisadas. Para cada `elseif`, uma nova expressão deve ser analisada. Quando todas as expressões forem falsas, então o último bloco `else` será executado.

```
<?php
    if ($a > $b) {
        echo "a é maior que b";
    } elseif ($a == $b) {
        echo "a é igual a b";
    } else {
        echo "a é menor que b";
    }
?>
```

O comando `switch` funciona semelhante a vários `if` juntos. Uma expressão ou variável é analisada e, de acordo com o valor, um entre vários blocos de comandos é executado. Diferentemente do `if`, cuja expressão somente retorna verdadeiro ou falso, no `switch` o valor retornado pode ser diverso (GLAZAR, 2011). A expressão é comparada com cada uma das cláusulas `case` até que uma coincida. Quando isso acontece, o bloco de comandos correspondente é executado até encontrar o comando `break`, que interrompe a execução daquele bloco e finaliza o `switch`. Se nenhuma cláusula coincidir, então o bloco delimitado pelo comando `default` é executado.

```
<?php
switch ($i) {
    case "apple":
        echo "i is apple";
        break;
    case "bar":
        echo "i is bar";
        break;
    case "cake":
        echo "i is cake";
break;
}
?>
```

10.3.2 Comandos de repetição

Os comandos de repetição servem para executar repetidas vezes o mesmo bloco de comandos, até que uma condição de parada seja atingida. Os comandos são: `while`, `do...while` e `for`. A diferença entre

eles está na condição de parada das repetições e no contador de iteração.

Já no `while` a condição de parada é testada no início da iteração. Se for verdadeira, repete o bloco de comandos; se for falsa, interrompe as repetições.

```
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

O comando `do...while` funciona de maneira semelhante ao `while`; a diferença é que a condição é testada depois do bloco de comandos. Isso garante que pelo menos uma vez o bloco de comandos será executado.

```
<?php
    $i = 0;
    do {
        echo $i;
    } while ($i > 0);
?>
```

O comando `for` é utilizado quando se conhece a quantidade total de iterações ou quando se pretende contar essas iterações. Sua sintaxe é `for (inicialização; condição; incremento) { .. }`

Onde :

- Inicialização – é uma instrução de atribuição executada apenas uma vez, no início do laço. Geralmente utilizada para inicializar a variável que irá controlar o número de repetições do laço.

- **Condição** – é a expressão que controla a parada das repetições. Se for verdadeira, o bloco de comandos é executado novamente; se for falsa, termina.
- **Incremento** – define a maneira como a variável de controle do laço será alterada a cada vez que o laço for repetido. Ela é executada ao final da execução de cada repetição do corpo do laço.

```
<?php
    /* exemplo 1 */

    for ($i = 1; $i <= 10; $i++) {
        echo $i;
    }
?>
```

10.4 ARRAY

Os arrays são estruturas para armazenar valores que precisam ser indexados. Diferentemente do Java, em que os índices são apenas números inteiros e consecutivos, em PHP os índices podem ser de vários tipos. Mesmo se forem inteiros não precisam ser consecutivos. Os valores armazenados não precisam ser do mesmo tipo. Veja no exemplo a seguir, onde `print_r` mostra todos os elementos do array.

```
<?php
    $a = array(
        NULL => 'zero',
        1     => 'one',
        2     => 'two');
```

```

var_dump(array_keys($a));
reset($a);
while( key($a) !== NULL )
{
    echo key($a) . ": ".current($a) .
"<br>"; // PHP_EOL
    next($a);
}
reset($a);
while( key($a) != NULL ) // '' == null
=> no iteration will be executed
{
    echo key($a) . ": ".current($a) .
"<br>"; // PHP_EOL
    next($a);
}
?>

```

10.5 FUNÇÕES

As funções em PHP seguem o mesmo princípio das de outras linguagens. A diferença é que como não precisamos declarar os tipos, a lista de parâmetros possui apenas o nome das variáveis. As funções que retornam valor também não precisam informar o tipo de retorno (GLAZAR, 2011). A sintaxe é:

```
function nome_função (lista de parâmetros) { ... }
```

Para organizar melhor o código, as declarações das funções ficam dentro do bloco <HEAD>, em uma página HTML. As chamadas das funções ficam no <BODY>.

```
<?php
```

```

function foo ($arg_1, $arg_2, /* ..., */
    $arg_n)
    {
        echo "Exemplo de função.\n";
        return $valor_retornado;
    }
?>

```

Por definição, a passagem de parâmetros é por valor. Caso se queira passar os parâmetros por referência, para alterar uma variável dentro da função, utiliza-se o símbolo '&' antes do parâmetro.

```

<?php
function foo(&$var){
    $var++;
}
$a=5;
foo($a);
// $a é 6 aqui
?>

```

10.6 FORMULÁRIO EM PHP

Vamos agora criar as páginas dinâmicas tratando as informações obtidas de outras páginas, como por exemplo, de formulários.

A partir de um formulário em HTML em um navegador, o usuário envia dados para uma página no servidor. Aprenderemos os comandos em PHP para capturar esses dados e fazer sua validação.

Na definição de um formulário em HTML, três atributos serão importantes para a criação de páginas em PHP no servidor: action, method e enctype. Vejamos um exemplo a seguir.

```

<form name="formteste"
      action="acao.php"
      method="post"
      enctype=application/x-www-form-urlencoded>
  <p>Nome: <input type="text" name="nome"
/></p>
  <p>Sobre Nome: <input type="text"
name="sobrenome" /></p>
</form>

```

Onde `name` é o nome do formulário e pode ser útil para referência em funções *javascript*. O `action` é o arquivo no servidor que será chamado para tratar os dados do formulário. A página em HTML com o formulário irá passar os dados para o programa PHP especificado por este campo. Já o `method` é, como o nome já diz, o método de envio de dados para o servidor. Dois tipos são permitidos: `GET` e `POST`, o quais detalharemos a seguir.

Por fim, o `enctype` que define o formato de como os dados serão enviados para o servidor. Por enquanto, nem precisamos especificar nada. O exemplo acima mostra o valor padrão, que pode ser omitido (GLAZAR, 2011).

A diferença entre `GET` e `POST` está na forma como os dados são enviados para o servidor. No método `POST`, os dados são enviados ocultos. É o método recomendado quando se utiliza formulário. Já no método `GET`, os dados são enviados de forma aberta, na URL, na forma do par ‘campo=valor’. Para isso, utiliza-se o símbolo ‘?’ depois do nome do arquivo. Os pares ‘campo=valor’ são separados pelo símbolo ‘&’. Essa forma de envio é utilizada quando temos poucas informações a serem passadas, e elas podem ser especificadas direto na URL. Utiliza-se principalmente quando queremos passar dados através de um link, sem os

campos de um formulário, como por exemplo em:

```
http://www.dev.com/cadastro.php?nome=Pedro&sobre-
nome=Machado
```

Nesse exemplo foi utilizado o método GET, que chama o programa cadastro.php, passando o campo nome igual a 'Pedro' e o sobrenome igual a 'Machado'.

Considerando o exemplo anterior, vimos que não há nada de especial nesta página. É um formulário HTML comum sem nenhuma tag especial de qualquer tipo. Quando o usuário preencher este formulário e clicar no botão enviar, a página action.php é chamada. Nesta página usaremos o comando `$_POST`, se o método de envio for o POST, e `$_GET`, se o método de envio foi o GET.

```
Olá <?php echo htmlspecialchars($_POST['nome']); ?>.
```

```
Seu sobre nome é <?php echo (int)$_POST['sobrenome']; ?>.
```

10.7 HEADERS

Os headers servem para o gerenciamento da conexão entre o PHP e o navegador do usuário, podendo trocar informações contidas no cabeçalho HTTP de uma página.

Os headers são informações trocadas entre o navegador e o servidor de maneira transparente ao usuário, e podem conter dados sobre o tipo e a versão do navegador, a página de onde partiu a requisição (link), os tipos de arquivos aceitos como resposta, e uma série de outras informações.

Por exemplo, o cabeçalho "HTTP/" indica que um código de retorno é enviado para o navegador do cliente. O exemplo a seguir mostra o envio de uma mensagem de "Página não encontrada":

```
<?php
    header("HTTP/1.0 404 Not Found");
?>
```

Uma segunda forma de usar o header é para redirecionar para outra página. Esse comando é muito útil para, ao final de uma página puramente em PHP, redirecionar para outra página padronizada de resposta ou de erro. Usa-se o comando “Location” e logo em seguida o nome da página que se pretende redirecionar.

```
<?php
    header('Location:      http://www.example.
com/');
    exit;
?>
```

O comando header deve ser usado antes de qualquer comando de exibição (echo, tags HTML, include).

10.8 ACESSO A BANCO DE DADOS

Como já foi dito anteriormente, é fundamental que se revise a linguagem SQL, estudada na disciplina de Banco de Dados, por ser essa a linguagem universal dos bancos de dados. É por meio dela que o PHP irá acessar com o banco de dados.

A maioria dos sites dinâmicos acessa algum banco de dados. Com o PHP podemos acessar diversos banco de dados, como o MySQL, PostgreSQL, Oracle, SQL Server, Firebird, Sysbase, Informix, SQLite e outros mais. Para os bancos de dados que o PHP não tem um módulo específico, podemos utilizar os drivers ODBC¹⁷.

Um dos bancos de dados mais utilizados com o PHP é o MySQL

¹⁷ ODBC é um padrão para acesso a sistemas gerenciadores de bancos de dados (SGBD). Este padrão define um conjunto de interfaces que permitem o uso de linguagens de programação capazes de utilizar estas interfaces, para ter acesso a uma vasta gama de bases de dados distintas sem a necessidade de codificar métodos de acesso especializados.

(sessão 9.1). O PHP possui um módulo específico para esse banco. Utilizaremos o MySQL em nossos exemplos por ser um banco simples de operar e utilizar pouco processamento e memória, em comparação com os outros.

10.8.1 Conectando ao banco de dados

Em uma página PHP, o primeiro passo é conectar com o banco de dados. Utilizaremos o comando `mysql_connect` para criar essa conexão (GLAZAR, 2011). Ele abre a conexão de uma página em PHP com o banco de dados. Sua sintaxe é

```
mysql_connect(servidor, usuário, senha)
```

Onde `servidor` – IP (ou hostname) é a porta do servidor em que está o banco de dados, no formato `servidor: porta`. Se o banco de dados estiver no mesmo computador, pode usar `localhost`. Se a porta não for informada, será utilizada a porta padrão, que no MySQL é a 3306. Os demais parâmetros são usuário e senha cadastrados no banco de dados.

Podem ocorrer alguns erros ao se tentar abrir a conexão. Os casos mais comuns são: não encontrar o servidor ou o usuário e a senha não terem permissão de acesso ao banco. Caso ocorra algum erro, o ideal seria mostrar uma mensagem de erro e interromper a execução da página, já que sem a conexão não poderemos acessar o banco de dados. Para isso usaremos a função `die`.

```
<?php
$link = mysql_connect('localhost', 'mysql_user',
'mysql_password');
if (!$link) {
    die('Não foi possível conectar: ' . mysql_er-
ror());
}
```

```

echo 'Conexão bem sucedida';
mysql_close($link);
?>

```

O `die` pode ser usado junto com outra função. Se der erro nessa função, o `die` é chamado automaticamente. Utiliza-se o operador `or` para associar o `die` a alguma função. O exemplo abaixo chama a função `mysql_connect`; se der erro, automaticamente o `die` é executado para mostrar a mensagem e interromper a execução da página.

```

mysql_connect(localhost,"root","root") or die("Erro ao
conectar. " .
mysql_error() );

```

Caso a abertura da conexão ocorra normalmente, o segundo passo é escolher o nome do banco de dados que será utilizado. Em um servidor de banco de dados, podem existir vários bancos (GLAZAR, 2011). Usaremos para isso a função `mysql_select_db`. Essa função seleciona o banco a ser utilizado no servidor conectado anteriormente. Sua sintaxe é `mysql_select_db("nome_banco")`.

Cada página que necessitar acessar o banco de dados deverá ter esses dois comandos no início. Como um sistema web geralmente possui várias páginas, replicar esses comandos não será uma boa solução. Caso tenha que mudar algum parâmetro, como por exemplo, o IP do servidor, todas as páginas sofrerão modificações.

Para evitar esse trabalho de manutenção, colocam-se os comandos de conexão com o banco em um único arquivo e todas as páginas fazem acesso a esse arquivo utilizando o comando `include` ou `include_once`. O `include` insere pedaços de códigos PHP de um determinado arquivo na página atual. Já o `include_once` verifica se o arquivo já foi inserido anteriormente, ou seja, insere somente uma única vez.

Sua sintaxe é:

```
include("nome_arquivo.php") ou
include_once("nome_arquivo.php")
```

Uma vez conectado com o banco de dados, podemos realizar todas as operações para manipulação dos dados: inserir, pesquisar, alterar e excluir. O que precisaremos saber para realizar essas operações é sobre a linguagem SQL. Uma conexão estabelecida com o comando `mysql_connect` é encerrada, automaticamente, ao final da execução da página. Caso queira encerrá-la antes disso, deve ser utilizado o comando `mysql_close` que fecha a conexão com o banco de dados. Sua sintaxe é: `mysql_close(identificador)` onde `identificador` é a variável que indica a conexão criada. Se o `identificador` não for fornecido, a última conexão estabelecida será encerrada.

10.8.2 Inserindo dados

A linguagem padrão de comunicação com os bancos de dados é a linguagem SQL. Para fazer com que o PHP execute os comandos SQL no banco de dados MySQL, utiliza-se a função `mysql_query` que executa um comando SQL no banco de dados MySQL. Retorna verdadeiro (`true`) em caso de sucesso e falso (`false`) caso contrário. Sua sintaxe é:

`mysql_query(comando, conexao)` onde `comando` é o comando na linguagem SQL, como: `INSERT`, `SELECT`, `UPDATE`, `DELETE`; e `conexao` é um parâmetro opcional que indica a conexão com o banco de dados. Se não for informada, utiliza a última conexão aberta. Portanto, para inserir os dados no banco, o comando em SQL que é utilizado é o `INSERT`.

Na sintaxe do `INSERT`, a parte do `VALUES` é em que passamos os valores para o banco de dados. É nesse ponto que usaremos as variáveis com os valores obtidos dos formulários. O código a seguir apresenta este comando.

```

    $sql = mysql_query("INSERT INTO cadastro(id, nome,
        sobrenome) VALUES ('', '$nome', '$sobrenome')");
    $result = mysql_query($sql);

```

10.8.3 Listando os dados

Após inserir os dados no banco, temos a possibilidade de recuperá-los e mostrá-los para o usuário. O comando em SQL que faz isso é o `SELECT`. O comando em PHP para recuperar os dados é o mesmo usado no inserir, o `mysql_query`. O que muda é o comando SQL passado para o banco, que agora é o `SELECT`.

```

<?php
    $result = mysql_query('SELECT * WHERE
1=1');
    if (!$result) {
        die('Invalid query: ' . mysql_er-
ror());
    }
?>

```

O retorno de um `SELECT` no banco é um conjunto de registros. Precisamos percorrer todos esses registros, pegando o primeiro, passando para o próximo, e assim por diante até ao último. O comando em PHP que faz isso é o `mysql_fetch_assoc`.

Esse comando retorna um registro de uma consulta e aponta para o próximo registro. Retorna `false` quando não existir mais registros, ou seja, quando chegar ao último. Retorna o registro em forma de array, em que os índices são os nomes das colunas da tabela no banco de dados. Sua sintaxe é:

```
mysql_fetch_assoc(identificador)
```

O identificador é o resultado do `SELECT`, ou seja, o conjunto de registros. Como o `mysql_fetch_assoc` retorna um registro por vez, podemos colocá-lo como condição de parada em um comando `while`, para percorrer todos os registros e parar quando chegar ao último (`false`). O registro retornado a cada laço do `while` é guardado em uma variável do tipo `array`. Cada posição do `array` é um campo da tabela. Para pegar o valor, usa-se como índice do `array` o nome do campo definido na tabela do banco, desta forma:

```
$variavel_registro["nome_campo"]
```

O nome do campo deve ser o mesmo usado no banco de dados. Letras maiúsculas são diferentes de minúsculas. Já para saber o total de registros retornados por uma consulta ao banco, usamos o comando `mysql_num_rows` que retorna a quantidade de registros da última consulta ao banco (GLAZAR, 2011). Sua sintaxe é:

```
mysql_num_rows(identificador);
```

Onde `identificador` é o resultado do `SELECT`, ou seja, o conjunto de registros. A seguir um exemplo onde são usados esses comandos:

```
<?php
```

```
$conn = mysql_connect("localhost", "mysql_user",
"mysql_password");
if (!$conn) {
    echo "Unable to connect to DB: " . mysql_er-
ror();
    exit;
}
if (!mysql_select_db("mydbname")) {
    echo "Unable to select mydbname: " . mysql_
error();
    exit;
}
```

```

$sql = "SELECT id as userid, fullname, userstatus
      FROM   sometable
      WHERE  userstatus = 1";
$result = mysql_query($sql);
if (!$result) {
    echo "Could not successfully run query ($sql)
      from DB: " . mysql_error();
    exit;
}
if (mysql_num_rows($result) == 0) {
    echo "No rows found, nothing to print so am
exiting";
    exit;
}
while ($row = mysql_fetch_assoc($result)) {
    echo $row["userid"];
    echo $row["fullname"];
    echo $row["userstatus"];
}
mysql_free_result($result);

?>

```

10.8.4 Consultando no banco de dados MySQL

Em muitas situações, listar todas as informações de uma tabela do banco de dados não é uma boa opção. Tabelas que possuem grande quantidade de dados aumentam o tráfego da internet para trazer todas essas informações para a página. Vamos aprender agora como filtrar os dados de uma consulta. Podemos realizar uma consulta no banco de dados escolhendo qualquer um dos campos para pesquisar, ou uma combinação deles. Por exemplo, podemos listar todos os clientes que nasceram em um

determinado mês, ou listar todos os funcionários com salário maior que um determinado valor, ou ainda, listar todas as vendas do último mês que ainda não foram pagas.

Para selecionar determinadas informações do banco de dados, usaremos a cláusula WHERE (visto no exemplo anterior) dentro do SELECT (GLAZAR, 2011). O que veremos aqui é como pegar os campos de um formulário para fornecer como limite para a consulta e exibir a resposta na mesma página.

```
$sql="SELECT * FROM tabela WHERE variavel='§variavel'";
```

10.8.5 Excluindo no banco de dados MySQL

Na grande maioria dos sistemas, não se usa muito excluir informações do banco de dados, porque isso faz com que a empresa perca o histórico dos dados. O que se faz é colocar um campo na tabela que informa o estado da informação como por exemplo: “ativo”, “inativo”, “em andamento”, “finalizado”, “aguardando resposta”, etc.

Em vez de excluir o cliente do banco de dados, a empresa pode apenas mudar seu estado para inativo, para que no futuro possa realizar alguma campanha para recuperar os clientes perdidos.

Mesmo não sendo muito utilizado, em algumas situações pode ser necessário excluir informações do banco. Então vamos aprender a fazer isso: para a operação de excluir do banco, mais uma vez o que muda é o comando SQL. Os comandos em PHP são os mesmos: abrir a conexão, selecionar o banco e executar o comando SQL.

Na maioria dos sistemas, ao excluir, é necessário conhecer pelo menos um dado da tabela. Dificilmente iremos excluir vários dados de uma vez. O ideal seria obter a informação que corresponde à chave primária da tabela, por ser uma informação única. Uma das formas de fazer isso é usar algum mecanismo de pesquisa para escolher a informação a ser pesquisada. O exemplo seguinte apaga todos os registros da tabela pessoas

em que o nome='Pedro'.

```
<?php
    $con = mysql_connect('localhost', 'banco',
'abc123') or
    die('Não foi possível conectar');
    mysql_select_db("my_db", $con);
    mysql_query("DELETE FROM pessoas WHERE nome =
'Pedro'");
    mysql_close($con);
?>
```

10.8.6 Alterando no banco de dados MySQL

A operação de alterar dados no banco segue o mesmo princípio que as demais informações. Novamente o que muda é o comando SQL passado para o banco.

Porém, o que torna uma página web um pouco mais trabalhosa para alterar dados, mas nem tanto, é que devemos ter as informações que serão alteradas. Essas informações podem estar dispostas de diversas formas em uma página. O mecanismo mais fácil e apropriado para isso é o formulário em HTML. Então, ao escolher uma determinada informação para alterar, vamos chamar uma página com o formulário já preenchido com as informações anteriores. O usuário altera no formulário os dados que deseja e depois os envia para outra página em PHP que atualizará o banco de dados.

```
<?php
    $con = mysql_connect('localhost', 'banco',
'abc123') or die('Não foi possível conectar');
    mysql_select_db("my_db", $con);
```

```

mysql_query("UPDATE pessoas SET idade = '36'
WHERE nome = 'Pedro'");
mysql_close($con);
?>

```

10.9 GERENCIANDO SESSÕES

O uso de sessões permitirá que informações sejam trocadas entre páginas web de uma mesma sessão. Com isso, será possível fazer a validação do usuário em uma página e verificar em todas as outras se o usuário foi autenticado ou não. E ainda mais, de posse do tipo do usuário (administrador, gerente, caixa, operador, etc.), é possível fazer controle de acesso, programando em cada página os tipos de usuários que podem acessá-las.

A cada página visitada por um usuário, uma nova conexão é criada pelo HTTP do navegador ao servidor. As informações da conexão anterior não são mantidas. Entretanto, em diversas situações é desejável que certas informações sejam armazenadas temporariamente entre uma página e outra. Como exemplo, pode-se citar a autenticação de login, que é feita em uma página e em todas as outras é necessário verificar se o usuário está logado, além de permitir obter os dados desse usuário em todas as páginas.

10.9.1 Manipulando uma sessão

Para criar uma sessão, usa-se o comando: `session_start()` ;

Após esse comando, um array, chamado `$_SESSION`, é criado para manipularmos as informações armazenadas. Com esse array é possível incluir uma nova variável, alterar as existentes e excluir uma ou todas as variáveis (GLAZAR, 2011). As páginas em que for preciso acessar informações do array `$_SESSION`, deve-se usar também o comando `ses-`

`session_start()`, para acessar a sessão criada.

Os dados armazenados em uma sessão são armazenados no array `$_SESSION`. O índice desse array, entre colchetes, é o nome da variável de sessão. Para criar uma nova variável na sessão, basta colocar seu nome como índice do array `$_SESSION`. O exemplo a seguir mostra a criação da sessão e a adição de três variáveis de sessão.

```
<?php
    session_start();
    echo 'Bem vindo à página inicial';
    $_SESSION['nome'] = 'Pedro';
    $_SESSION['Sobrenome'] = 'Machado';
    $_SESSION['idade'] = 'nd';
?>
```

Para eliminar uma variável de uma sessão, utiliza-se o comando: `unset($_SESSION["nome_da_variável"])`. Uma sessão é automaticamente eliminada quando o navegador do usuário é fechado. Caso queira destruir a sessão antes disso, como por exemplo, ao clicar em um botão do tipo “*sair*” ou “*logout*”, devem ser realizadas três etapas: a primeira é acessar a sessão com o comando `session_start()`; a segunda é a liberação de todas as variáveis da sessão com o comando `session_unset()`; a última é a destruição da sessão com o comando `session_destroy()`. O programa a seguir mostra o exemplo para finalizar a sessão.

```
<?php
    // Inicializa a sessão.
    session_start();
    // Libera as variáveis da sessão
    session_unset();
```



```
// Por último, destrói a sessão  
session_destroy();  
?>
```

11 LINGUAGENS DE SCRIPT *CLIENT-SIDE* - JAVASCRIPT

A linguagem de cliente ou client-side scripting é uma linguagem que é executada no lado cliente, ou seja, no computador do próprio usuário, e por isso é usada nas situações em que a linguagem server-side não tem alcance. Entre as linguagens client-side, há o *JavaScript*, que é a única a qual realmente roda no navegador do usuário. Através do JavaScript é possível manipular a página do usuário diretamente, fazendo coisas dinâmicas que vão desde mudar o valor de um campo do formulário até criar uma área redimensionável que pode ser arrastada pela página.

11.1 LINGUAGENS DE SCRIPT

JavaScript é uma linguagem de script incorporada a um documento HTML. Historicamente, trata-se de uma das primeiras linguagens de scripts para a web. Esta linguagem é uma linguagem que permite a execução de comandos do cliente, ou seja, em termos do navegador e não do servidor web.

Deste modo, a linguagem *JavaScript* é altamente dependente do navegador que chama a página web onde o script está incorporado, mas, por outro lado, não requer nenhum compilador, ao contrário da linguagem Java, com a qual ela tem sido confundida, há muito tempo.

O *JavaScript* foi desenvolvido pela Netscape em 1995. Originalmente, era chamado LiveScript e fornecia a uma linguagem de script simples para o navegador Netscape Navigator 2. Ele foi criticado durante muito tempo pela sua falta de segurança, seu pouco desenvolvimento e pela ausência de alertas de erro explícitos tornando seu uso difícil. Em

1995, após uma associação com o fabricante Sun, Netscape renomeou sua linguagem Javascript. Na mesma época, a Microsoft desenvolveu a linguagem Jscript, uma linguagem de script muito semelhante. Assim, para evitar excessos de ambos os lados, uma norma foi definida para padronizar as linguagens de script, trata-se o ECMA 262, criado pela organização do mesmo nome (*European Computer Manufacturers Association*).

É importante que não se confunda JavaScript com Java. Na verdade, ao contrário da linguagem Java, o código é escrito diretamente na página HTML, que não permite nenhuma privacidade no que se refere aos códigos (eles são realmente visíveis). Na tabela a seguir vemos as principais diferenças entre as duas linguagens.

Tabela 4: Diferenças entre Javascript e Java

Javascript	Java
Linguagem interpretada	Linguagem pseudocompilada (download de uma máquina virtual)
Código integrado ao HTML	Código (applet) fora do documento HTML, chamado a partir da página
Linguagem flexível	Linguagem rígida (declaração do tipo da variável)
Ligações dinâmicas: as referências dos objetos são verificadas durante o download	Ligações estáticas: os objetos devem existir durante o download (compilação)
Acessibilidade do código	Privacidade do código
Seguro: não pode gravar no disco rígido	Seguro: não pode gravar no disco rígido

11.2 CONCEITOS BÁSICOS

Em documentos HTML, a utilização da linguagem JavaScript se dá sob a forma de funções(applets), as quais são chamadas em determinadas situações ou em resposta a determinados eventos, estas funções podem estar localizadas em qualquer parte

do código HTML (GONÇALVES (b), 2005). A única restrição é que devem começar com a declaração `<SCRIPT>` e terminar com o respectivo `</SCRIPT>`. Por convenção costuma-se colocar todas as funções no início do documento uma entre as TAGs `<HEAD>` e `</HEAD>`, isso para garantir que o código JavaScript seja carregado antes que o usuário interaja com a página, ou seja, antes do `<BODY>`. O código a seguir mostra um exemplo da utilização.

```
<HTML>
  <HEAD>
    <TITLE>Exemplo</TITLE>
  </HEAD>
  <BODY>
    Esta linha está escrita em HTML
    <SCRIPT>
      document.write("Aqui já é JavaScript");
    </SCRIPT>
    Voltamos para o HTML
  </BODY>
</HTML>
```

É importante ressaltar que todas as linhas devem ser terminadas com ; (ponto e vírgula) a menos que a próxima instrução seja um “else” e se você precisar escrever mais de uma linha para executar uma condição seja ela em uma estrutura “for”, “if” ou “while”, este bloco de instruções deve estar entre “{ }”(chaves). Inclusive a definição de funções segue este modelo, ou seja, todo o código da função deve estar limitado por { (no início) e } (no final).

11.3 VARIÁVEIS

Em *JavaScript*, variáveis dinâmicas podem ser criadas e inicializadas sem declarações formais. Existem dois tipos de abrangência para as variáveis:

- “Global” - Declaradas/criadas fora de uma função. As variáveis globais podem ser acessadas em qualquer parte do programa.
- “Local” - Declaradas/criadas dentro de uma função. Só podem ser utilizadas dentro da função para a qual foram criadas e precisa ser definida com a instrução `Var`.

Com relação à nomenclatura, as variáveis devem começar por uma letra ou pelo caractere sublinhado “_”, o restante da definição do nome pode conter qualquer letra ou número. É importante ressaltar que a variável “Código” é diferente da variável “código”, que por sua vez é diferente de “CODIGO”, sendo assim, muito cuidado quando for definir o nome das variáveis, utilize sempre um mesmo padrão (GONÇALVES (b), 2005).

Existem três tipos de variáveis: Numéricas, Booleanas e Strings. Como já era de se esperar, as variáveis numéricas são assim chamadas, pois armazenam números, as Booleanas valores lógicos (True/False) e as Strings, sequência de caracteres. As strings podem ser delimitadas por aspas simples ou duplas, a única restrição é que se a delimitação começar com as aspas simples, deve terminar com aspas simples, da mesma forma para as aspas duplas.

O *JavaScript* reconhece ainda um outro tipo de conteúdo em variáveis, que é o *NULL*. Na prática isso é utilizado para a manipulação de variáveis não inicializadas sem que ocorra um erro no seu programa. Quando uma variável possui o valor *NULL*, significa dizer que ela possui um valor desconhecido ou nulo. A representação literal para *NULL* é a

string 'null' sem os delimitadores. Quando referenciado por uma função ou comando de tela, será assim que *NULL* será representado. Observe que *NULL* é uma palavra reservada. Pode-se trabalhar ainda com Arrays da mesma forma que o Java (capítulo 6). Além disso, o Javascript oferece a mesma sintaxe do Java, inclusive a manipulação de objetos assim como o Java.

11.4 FUNÇÕES INTERNAS

A linguagem *JavaScript* além dos recursos descritos anteriormente, ainda possui algumas funções internas, que não estão ligadas diretamente a nenhum objeto, porém isso não impede que essas funções recebam objetos como parâmetros. A seguir estas funções serão vistas detalhadamente.

`alert` mostra uma caixa de alerta, seguido de um sinal sonoro e o botão de OK. Exemplo: `alert('Esta é uma janela de alerta!')`. Já o `confirm` mostra uma caixa de diálogo, seguida de um sinal sonoro e botão de OK e Cancel. Retorna um valor verdadeiro se o usuário escolher OK, como por exemplo, `retorno=confirm('Deseja prosseguir?')`.

O `escape` obtém o código ASCII de um caracter que não seja alfa-numérico. Sua sintaxe é `document.write(escape("@"))`. O `eval` avalia uma expressão numérica, retornando um resultado também numérico. Exemplo: `document.write(eval(10*9*8*7*6*5*4*3*2))`. Já o `parseFloat` converte uma string que representa um número, para um número com ponto flutuante. Caso a string não possa ser avaliada, a função retorna 0. Seu uso é `document.write(parseFloat("-32.465e12"))`. De forma análoga, o `parseInt` converte uma string, que representa um número em uma base predefinida para base 10. Caso a string possua um caracter que não possa ser convertido, a operação para, retornando o valor antes do erro (GONÇALVES (b), 2005).

11.5 OBJETOS JAVASCRIPT

JavaScript organiza todos os elementos de uma Home Page dentro de uma hierarquia. Cada elemento é visto como um objeto. Os objetos podem ter propriedades, métodos e responder a certos eventos. Por isso é muito importante entender a hierarquia dos objetos HTML.

Dentro da janela do *browser*, nós podemos carregar uma página HTML. Esta página é um *objeto document*. Desta forma, o *objeto document* representa o documento HTML (que está carregado no momento). O *objeto document* é muito importante, em *JavaScript* você sempre o usará muito. As propriedades e métodos do objeto *document* serão vistas detalhadamente, mais adiante.

Mas o que é mais importante é que todos os objetos HTML são propriedades do *objeto document*. Um objeto HTML pode ser, por exemplo, um link ou um formulário. Podemos obter informações de diversos objetos e então manipulá-los. Para isso nós devemos saber como acessar os diferentes objetos.

Verifica-se, primeiramente, o nome do objeto no diagrama de hierarquia. Caso seja necessário) saber como referenciar a primeira imagem na página HTML, basta seguir o caminho hierárquico. Deve-se percorrer o diagrama de cima para baixo, o primeiro objeto é chamado *document*, a primeira imagem é representada por *Imagem[0]*. Desta forma nós podemos acessar este objeto em JavaScript, da seguinte forma: `document.Imagem[0]`.

Caso queira saber o que o usuário digitou dentro do primeiro elemento do formulário, primeiro precisa-se pensar em como acessar esse objeto. Novamente seguiremos o diagrama de hierarquia, de cima para baixo indo até *Elem[0]*. Todos os nomes de objeto por onde se passou tem que constar na referência ao primeiro elemento do formulário. Desta forma pode-se acessar o primeiro elemento texto assim: `document.Form[0].Elem[0]`.

Para recuperar o texto digitado, usaremos uma propriedade do elemento texto chamada *value*. Outras propriedades, métodos ou eventos serão vistos detalhadamente mais adiante. O *value* armazena o conteúdo do objeto, ou seja, o texto digitado. A seguinte linha de código obtém o texto digitado: `nome = document.forms[0].elements[0].value;`

A *string* é armazenada na variável *name*. Nós podemos agora trabalhar com esta variável. Por exemplo, podemos criar uma janela popup com `alert("Oi "+name);`. Se a entrada for “Pedro” o comando `alert("Oi "+name)` abrirá uma janela popup com o texto “Oi Pedro”.

Se estiver trabalhando com páginas muito grandes pode ficar um pouco confuso referenciar objetos diretamente pelo endereçamento do diagrama de hierarquia. Neste caso pode se ter referências do tipo: `document.forms[3].elements[15]` ou `document.forms[2].element[21]`. Para evitar esse problema deve-se dar nomes diferentes aos objetos, vejamos o seguinte fragmento de um documento HTML:

```
<form NAME="form">
Nome: <input TYPE="text" NAME="sobrenome" value=" ">
```

...

Dessa forma, em vez de usarmos, por exemplo:

```
document.forms[0].elements[0].value;
```

Podemos usar:

```
document.form.sobrenome.value;
```

Podemos usar:

```
document.form.sobrenome.value;
```

Isto traz muitas facilidades, especialmente com páginas grandes e

com muitos objetos. Observe que a caixa alta das letras faz diferença. Muitas propriedades dos objetos *JavaScript* não são apenas para leitura, pode-se atribuir novos valores a algumas propriedades. Observe o exemplo o *Location*. Este objeto contém informações sobre a URL da página atual e sua forma geral é `location.propriedade` ou `location.metodo()`.

A propriedade `hash` funciona de forma semelhante ao famigerado “go to” de algumas linguagens de programação (GONÇALVES (b), 2005). Normalmente é usado em links que acessam a mesma página. O exemplo abaixo demonstra a utilização da propriedade `hash`, para criar um link para outro trecho da mesma página.

```
<HTML>
    ...
    <A HREF = "location.hash='2'">Item 1</A>
    ...
    <A NAME = "1"></A>Item 1
    ...
    <A NAME = "2"></A>Item 2
    ...
</HTML>
```

A propriedade `host` armazena uma string com o formato “host-name:port” da página HTML atual. Ex: `alert('Demonstração da propriedade host: '+location.host)`. Já o `hostname` armazena uma string com o IP da página HTML atual. Um exemplo de uso é `alert('Demonstração da propriedade host: '+location.host)`. O método `href` identifica a url mostrada na barra “location” do browser. Ex: `alert('A URL desta página é: '+ location.href)`. A propriedade *pathname* contém uma *string* com o path da página HTML atual. Ex: `alert('O path da URL desta página é: '+ location.pathname)`.

11.5.1 Button

Este objeto nada mais é que um botão 3-D (no mesmo padrão do Windows) que ao ser pressionado produz uma sensação de profundidade e geralmente chama uma função. Pode ser utilizado para inúmeras aplicações, dependendo apenas de sua imaginação, a única precaução é defini-lo dentro de um formulário:

```
<Input Type="button" Name="NomeDoBotão" Value="Rótulo"
onClick="RespostaAoEvento">
```

Onde:

- Type - define o objeto
- Name - define o nome do objeto para nossa aplicação. É por este nome que referenciamos alguma propriedade deste objeto.
- Value - define o que será escrito na face do botão.
- onClick - É o único evento possível para este objeto, normalmente define uma função a ser executada quando clicamos no botão.

Propriedades:

- NAME: Informa o nome do objeto button em forma de string, da mesma forma como definido no campo Name que foi utilizado na definição do botão. É importante não confundir o campo Name com a propriedade NAME, veja a diferença:

```
<input type="button" name="OK" value="-
Confirma" onClick="ConfirmaInformacoes()">
OK.Name // equivale a "OK"
```

- VALUE: Informa o label do botão em forma de string da mesma forma como foi definido no campo Value que foi utilizado na definição do botão.

```
OK.Value // equivale a "Confirma"
```

Métodos:

- `click`: este método simula um clique do mouse no objeto `Button` (itálico), ou seja, executa um procedimento associado a um botão como se o botão tivesse sido pressionado, mas sem que o usuário tenha realmente clicado.

```
OK.click() // executaria a função ConfirmaInformacoes
```

Eventos associados:

`onClick`: Define o que fazer quando clicamos no objeto `Button`. O exemplo a seguir mostra o uso desse evento.

```
<FORM>
<INPUT TYPE="button" VALUE="Clique aqui!!!"
NAME="botaol" onClick="alert('Apropriedade
NAME deste botão é:'+botaol.name+'\nA pro-
priedade VALUE deste botãoé:'+botaol.val-
ue)')">
</FORM>
```

11.5.2 Navigator

Neste objeto ficam armazenadas as informações sobre o browser que está sendo utilizado. Forma geral:

```
Navigator.propriedade
```

Propriedades:

`appName` - Armazena o codnome do browser.

Ex: `Navigator.appCodeName`

`appName` - Armazena o nome do browser.

Ex: `Navigator.appName`

`appVersion` - Armazena a versão do browser

Ex: `Navigator.appVersion`

`userAgent` - Armazena o cabeçalho (user-agent) que é enviado para o servidor, no protocolo HTTP, isto serve para que o servidor identifique o software que está sendo usado pelo cliente.

Ex: `Navigator.userAgent`

11.5.3 Form

Os formulários têm muitas utilidades, uma das principais seria a transferência de dados dentro da própria página HTML, sem que para isso seja necessária a intervenção de qualquer outro meio externo (GONÇALVES (b), 2005).

Ao criarmos um formulário na página HTML, automaticamente é criada uma referência para este formulário, que fica guardada na propriedade `form` do objeto `document`. Como foi visto na página que trata do objeto `document`, a propriedade `form` é um vetor, e a cada formulário criado também é criado um novo elemento para este vetor, o índice inicial deste vetor é 0 (zero) e varia até o número de formulários do documento -1.

Pode-se notar que cada formulário criado em uma página HTML é considerado um objeto distinto, tendo suas próprias referências, métodos, propriedades e eventos associados. A forma de acessarmos diferentemente esses formulários dentro da página HTML é utilizar a propriedade `form` do objeto do documento. Forma geral:

```
<FORM NAME="Nome" ]
[ ACTION="URL" ]
[ METHOD="GET | POST" ]
```

```
[onSubmit="evento"]>
```

Onde:

- Nome = Nome do formulário, para futuras referências dentro da página HTML.
- URL = Especifica o URL do servidor ao qual será enviado o formulário.
- GET | POST = métodos de transferência de dados do browser para o servidor

Propriedades:

- action - Especifica o URL do servidor ao qual será enviado o formulário.

Ex: `document.NomeDoFormulario.action`

```
document.GuestBook.action = "esaex@canudos.ufba.br"
```

- elements - Vetor que armazena todos os objetos que são definidos dentro de um formulário (caixas de texto, botões, caixas de entrada de dados, checkboxes, botões de rádio). O número de elementos deste vetor varia de 0 (zero) até o número de objetos dentro do formulário -1.

Ex: `document.NomeDoFormulario.elements`

```
[indice]
```

0 method - Seleciona um método para acessar o URL de ação. Os métodos são: get e post. Ambos os métodos transferem dados do browser para o servidor, com a seguinte diferença:

0 method=get - os dados de entrada são acrescentados ao endereço (URL) associado, como se fossem uma query (pesquisa a banco de dados) comum;

0 `method=post` - os dados não são acrescentados ao URL, mas fazem parte do corpo da mensagem enviada ao servidor.

Esta propriedade pode ser alterada, porém só surtirá efeito antes que o formulário seja mostrado na tela.

```
Ex: document.NomeDoFormulario.method = "post"
( ou "get" )
```

O método `submit` transfere os dados do formulário para o endereço especificado em `action`, para serem processados. Funcionando de maneira análoga ao botão `submit` em HTML. A sintaxe do comando é `document.NomeDoFormulario.submit()`.

Como evento temos o `onSubmit` que ocorre quando um botão do tipo `submit` recebe o clique do mouse, transferindo os dados de um formulário para o servidor especificado em `action`. Os dados só são enviados se o evento receber um valor verdadeiro (`true`), valor este que pode ser conseguido como resultado a chamada de uma função que valida as informações do formulário (GONÇALVES (b), 2005). Exemplo:

```
document.NomeDoFormulario.onSubmit="return valida_Informacoes(NomeDoFormulario)"
```

11.5.4 CheckBox

Este objeto como o próprio nome sugere, exibe uma caixa de checagem igual às que encontramos no Windows, o funcionamento também é o mesmo: a condição de selecionada ou não é alternada quando clicamos o mouse sobre o objeto, ou seja, se clicarmos sobre um objeto *checkbox* já marcado ele será automaticamente desmarcado, ao passo que se clicarmos em um objeto *checkbox* desmarcado ele será automaticamente marcado. O código exemplo a seguir mostra o uso e as propriedades do *checkbox*.

Forma geral:

```

<FORM>
    <INPUT    TYPE="checkbox"    NAME="NomeDo-
Objeto"    [CHECKED]    VALUE="Label"    on-
Click="Ação">
</FORM>

```

Onde:

- Type - Nome do objeto;
- Name - Nome dado pelo programador, para futuras referências ao objeto;
- CHECKED - Se especificado a CheckBox já vai aparecer selecionada;
- Value - Define um rótulo para a CheckBox.
- onClick - Define o que fazer quando se dá um clique na CheckBox, fazendo com que o objeto CheckBox funcione como um objeto Button.

Propriedades:

- name - Nome do objeto CheckBox em forma de string, da mesma forma como definido no campo Name utilizado na criação da CheckBox.

```
NomeDoObjeto.name // equivale a string "NomeDoObjeto"
```

- value - Armazena o conteúdo do campo VALUE, definido na TAG.

```
NomeDoObjeto.value
```

- checked - Retorna um valor lógico que depende do estado do objeto CheckBox

```
NomeDoObjeto.checked
```

- // equivale a True se o objeto selecionado e False caso

contrário

- `defaultChecked` - Informa/Altera o estado default de um objeto `CheckBox`. Com relação a alteração, somente os objetos `CheckBox` ainda não exibidos podem ter seu estado default alterado.

`NomeDoObjeto.defaultChecked//` equivalerá a `True`, se a cláusula `CHECKED` estiver presente e a `False` caso contrário

- Métodos:
 - `click`: este método simula um clique do mouse no objeto `CheckBox`, ou seja, executa um procedimento associado a uma `CheckBox` como se estivéssemos clicado na `CheckBox`, mas sem que o usuário tenha realmente clicado.

`Select01.click()` // executaria uma função

- Eventos associados:
 - `onClick`: Define o que fazer quando clicamos no objeto `CheckBox`

Exemplo:

```
<HTML>
  <HEAD>
    <TITLE>Exemplo CheckBox</TITLE>
    <SCRIPT>
      function exemplo(p1,p2,p3,p4)
      {
        alert('Veja os conteúdos das
propriedades:
  \nName='+p1+
  '\nValue='+p2+
```

```

        '\nChecked=' +p3+
        '\ndefaultChecked=' +p4);
    }
</SCRIPT>
...

```

11.5.5 Document

Este objeto armazena todas as características da página HTML, como por exemplo: cor das letras, cor de fundo, figura que aparecerá como papel de parede, etc. Sempre que incluímos alguma declaração no <body> do documento, estamos alterando o objeto Document. Sua forma geral é:

```

<body [background="imagem"]
      [bgcolor="#cordefundo"]
      [fgcolor="#cordotexto"]
      [link="#cordoslinks"]
      [alink="#cordolinkativado"]
      [vlink="#cordolinkvisitado"]
      [onload="função"]
      [onunload="funcao"]>

```

Onde:

- Imagem = figura no formato GIF, que servirá como papel de parede para a Home Page;
- #Cor... = número (hexadecimal), com seis dígitos, que corresponde à cor no formato RGB, o “#” é obrigatório. Os dois primeiros dígitos correspondem à R (red), os dois do meio a G (green) e os dois últimos a B (blue). A combinação dos três, forma a cor no formato RGB.
- função = Nome de uma função predefinida, que será

chamada quando o evento ocorrer.

Propriedades:

- *alinkColor* - Determina a cor do link enquanto o botão do mouse estiver pressionado sobre ele. Exemplo: `document.alinkColor="#FFFFFF"`
- *anchors* - Vetor que armazena as âncoras definidas em uma página HTML com o comando ``. Esta propriedade é somente para leitura, não pode ser alterada. `document.anchors[índice]`
- *bgColor* - Determina a cor de fundo da página HTML. `document.bgColor="#000000"`
- *cookie* - Os cookies são pequenos arquivos que alguns sites da Web gravam no computador dos visitantes. A idéia é identificar o usuário, anotar quais caminhos ele já percorreu dentro do site e permitir um controle mais eficaz dos espectadores.
- *fgColor* - Determina a cor das letras em uma página HTML. Esta propriedade não altera o que já está impresso na página HTML. `document.fgColor="#0000FF"`
- *forms* - Vetor que armazena as referências aos formulários existentes na página HTML. Esta propriedade é somente para leitura, não pode ser alterada. Ex: `document.forms[índice]`
- *lastModified* - Obtém a data da última atualização da página HTML. Esta propriedade é somente para leitura, não pode ser alterada. `document.lastModified`
- *linkColor* - Determina a cor dos links que ainda não foram visitados pelo usuário. Ex: `document.linkColor="#00FF00"`
- *links* - Vetor que armazena os links definidos em uma página HTML. Esta propriedade é somente para leitura,

não pode ser alterada. `document.links[índice]`

- *location* - Armazena o endereço (URL) atual em forma de string. Esta propriedade é somente para leitura, não pode ser alterada.
- *referrer* - Armazena o endereço (URL) de quem chamou a página HTML atual. Com essa propriedade pode-se saber como usuário chegou à sua página. Ela é somente para leitura, não pode ser alterada. `document.referrer`
- *title* - Armazena uma string com o título da página HTML atual. Esta propriedade é somente para leitura, não pode ser alterada. `document.title`
- *vlinkColor* - Determina a cor que o link aparecerá após ser visitado. Ex: `document.vlinkColor = "#80FF80"`
- Métodos:
- *clear* - limpa a tela da janela atual. `document.clear()`
- *open* - Abre um documento e envia (mas não exibe) a saída dos métodos `write/writeln`. Os dados enviados são exibidos quando é encontrado o método `close`. `document.open()`
- *close* - Termina uma sequência iniciada com o método `open`, exibindo o que tinha sido apenas enviado. Ex: `document.close()`
- *write* - Imprime informações na página HTML. `document.write("Qualquer coisa" [,variável] [,..., expressão])`
- *writeln* - Imprime informações na página HTML e passa para a próxima linha. Ex: `document.writeln("Qualquer coisa" [,variável] [,..., expressão])`
- Eventos:
- *onLoad* - Ocorre assim que um browser carrega uma página HTML ou frame. Ex: `<BODY ... onLoad='alert("Oi!!!")'>`

- `onUnload` - Ocorre quando se abandona uma página HTML ou frame. `<BODY ... onUnload='alert("Tchau!!!")'>`

11.5.6 History

Este objeto armazena todas as URL das páginas HTML por onde o usuário passou durante a sessão atual do browser (GONÇALVES (b), 2005). É uma cópia das informações armazenadas na opção “Ir” da barra de menu do browser.

Forma geral:

```
history.propriedade
history.método
```

Propriedades:

- `length` - Informa a quantidade de páginas visitadas. Ex: `history.length`
-
- Métodos:
- `back` - Retorna à página anterior, de acordo com a relação de páginas do objeto `history`. Equivale a clicar o botão `back` do browser. `history.back()`
- `forward` - Passa para a próxima página, de acordo com a relação de páginas do objeto `history`. Equivale a clicar o botão `forward` do browser. `history.forward()`
- `go` - Permite que qualquer URL que esteja presente na relação de páginas visitadas do objeto `history`, seja carregada. `history.go(parâmetro)`

Existem duas possibilidades para “parâmetro”. Ele pode ser um número. Ao definir o número, este deve ser inteiro. Se for positivo, a página

alvo está “parâmetro” páginas à frente. Ao passo que se for negativo, a página alvo está “parâmetro” páginas para atrás. Ele também pode ser uma string. Neste caso, o alvo é a URL que mais se assemelhe ao valor da string definida por “parâmetro”.

11.5.7 Window

É o objeto que ocupa o topo do esquema hierárquico em JavaScript. Ele contém todos os outros objetos.

Propriedades:

- `defaultStatus` - Determina o conteúdo padrão da barra de status do browser, quando nada de importante estiver acontecendo. `window.defaultStatus='Qualquer coisa'`
- `frames` - Vetor que armazena as referências para as frames da janela atual. Ex: `parent.frames.length` // obtém o número de frames da janela principal, assumindo que estamos em uma frame.
- `parent` - Refere-se à janela pai da frame atual.
- `self` - Refere-se à janela atual. Ex: `self.defaultStatus='Qualquer coisa'`
- `status` - Define uma mensagem que irá aparecer no rodapé do browser, em substituição, por exemplo, a URL de um link, quando estivermos com o mouse sobre o link. `window.status="qualquer texto"`
- `top` - Refere-se à janela de nível mais alto do esquema hierárquico do JavaScript. `top.close()` // fecha a janela principal do browser
- `window` - Refere-se à janela atual. Funciona de modo análogo a `self`. `window.status='Qualquer coisa'`
- Métodos:
- `alert` - Mostra uma caixa de alerta, seguido de um sinal

sonoro e o botão de OK. Ex: alert(‘Esta é uma janela de alerta!’)

- close - Termina a sessão atual do browser. top.close()
- confirm - Mostra uma caixa de diálogo, seguida de um sinal sonoro e os botões de OK e Cancel. Retorna um valor verdadeiro se o usuário escolher OK. Ex: retorno=confirm(‘Deseja prosseguir?’)
- open – Abre uma nova sessão do browser, como se o usuário pressionasse <CTRL>+N Ex: window.open(“URL”, “Nome” [, “características”])

Onde:

- URL : endereço selecionado inicialmente quando da abertura da nova janela.
- Nome : nome da nova janela, definido pelo programador;
- Características - série de opções de configuração da nova janela, se especificados devem estar na mesma string, separados por vírgulas e sem conter espaços:
 - toolbar=0 ou 1
 - location=0 ou 1
 - directories=0 ou 1
 - status=0 ou 1
 - menubar=0 ou 1
 - scrollbars=0 ou 1
 - resizable=0 ou 1
 - width=valor inteiro positivo
 - height=valor inteiro positivo
 - window.open(“http://www.ufpi.br”, “NovaJanela”, “toolbar=1,location=1,directo-

- `ries=0,status=1,menubar=1,`
- `scrollbars=1,resizable=0,width=320,height=240”)`
- `prompt` - Monta uma caixa de entrada de dados, de forma simplificada comparando-se como objeto `text`. Ex: `prompt(label [,valor])`
- onde:
- `label` : texto que aparece ao lado da caixa;
- `valor` : é o conteúdo inicial da caixa;\
- `setTimeout` - Faz com que uma expressão seja avaliada após um determinado tempo (em milissegundos).
- Ex: `ID=setTimeout(alert(‘você chegou aqui, a 10 segundos’),10000)`
- `ID` - identificador utilizado para o cancelamento de `setTimeout`
- `clearTimeout` - Cancela `setTimeout`. Ex: `clearTimeout(ID)`

Eventos:

- `onLoad` : Ocorre assim que a página HTML termina de ser carregada.
- `onUnload` : Ocorre assim que o usuário sai da página atual.

11.5.8 Link

O HTML permite ligações de uma região de texto (ou imagem) a outro documento. Nestas páginas, temos visto exemplos dessas ligações: o browser destaca essas regiões e imagens do texto, indicando que são ligações de hipertexto - também chamadas `hypertext links` ou `hyperlinks` ou simplesmente `links`.

Forma geral:

```
<A [ NAME="ancora" ]
      HREF="URL"      [TARGET="janela" ]      [on-
Click="ação"] [onMouseOver="ação"]>
Texto explicativ
o</A>
```

onde:

- URL: é o documento destino da ligação hipertexto;
- âncora: é o texto ou imagem que servirá de ligação hipertexto .
- janela : Nome da janela onde a página será carregada, para o caso de estarmos trabalhando com frames:
- “_top”: Se estivermos trabalhando com frames, a página referenciada pelo link, substituirá a página que criou as frames, ou seja, a página atual, com todas as frames, dará lugar a nova página.
- “_self”: a nova página é carregada na mesma janela do link.
- “_blank”: Abre uma nova seção do browser para carregar a página.
- ação: Código de resposta ao evento.
- Texto explicativo: Texto definido pelo usuário, que aparece na tela de forma destacada.

Eventos:

- onClick - Ocorre quando clicamos o mouse sobre o link. Ex: Texto
- onMouseOver - Ocorre quando o mouse pas-

sa por cima do link, sem ser clicado. Ex: Texto

Existem três caminhos para o documento destino. O primeiro deles é o caminho relativo que pode ser usado sempre que queremos fazer referência a um documento que esteja no mesmo servidor do documento atual. Para usar links com caminhos relativos é preciso, portanto, conhecer a estrutura do diretório do servidor no qual estamos trabalhando. Simplificando, é como acessarmos um arquivo que esteja no mesmo diretório, não sendo necessário acrescentar o path.

Já o caminho absoluto é utilizado quando desejamos referenciar um documento que esteja em outro servidor. Com a mesma sintaxe, é possível escrever links para qualquer servidor WWW no mundo. Seria a aplicação do endereço completo da página em questão, como visto no ítem anterior (GONÇALVES (b), 2005).

Por fim, temos as ligações a trechos de documentos. Além do atributo href, que indica um documento destino de uma ligação hipertexto, o elemento A possui um atributo NAME que permite indicar um trecho de documento como ponto de chegada de uma ligação hipertexto. Funciona tanto para documentos locais como para os armazenados em outro servidor. O trecho deve ser indicado com o símbolo "#".

EXERCÍCIOS

As questões abaixo devem ser respondidas em forma dissertativa e argumentativa com pelo menos uma lauda. Devem também refletir a interpretação da leitura do texto juntamente com pesquisas sobre o tema arguido.

1. Quais os pré-requisitos que um servidor deve dispor para a execução do PHP. Explique cada um deles.
2. Quais as diferenças entre uma linguagem de script do lado do cliente e do lado servidor?
3. O que é uma sessão em um ambiente de desenvolvimento web?
4. (Questão Prática) Quais as diferenças entre os comandos de repetição `while`, `do...while` e `for`? Crie um exemplo em PHP para cada um que mostre essas diferenças.
5. Qual a diferença entre os métodos GET e POST na submissão de um formulário PHP?
6. Qual o principal uso do JavaScript? Explique porque ele é diferente do JSP.
7. (Questão Prática) Escreva uma página em PHP que contenha um formulário com uma série de campos para cadastro de clientes e que submeta esse formulário a gravação em um banco de dados com os mesmos campos. Escreva também códigos JavaScript que impeçam que campos sem conteúdo sejam enviados ao banco de dados.

WEBLIOGRAFIA

Apostila de PHP

<http://www.abruem.org.br/uploads/foruns/56/palestras/php.pdf>

Curso de Linguagem PHP

<http://www.etelg.com.br/paginaete/downloads/informatica/php.pdf>

Desenvolvendo aplicações com PHP e MySQL

https://danielmoreira.files.wordpress.com/2010/08/guia_php.pdf

JavaScript

<http://www.cybertechse.com.br/Documents/Linguagens%20de%20marca%C3%A7%C3%A3o/Apostilas/AT%2004%20-%20Apostila%20JavaScript.pdf>

Desenvolvimento Web com HTML, CSS e JavaScript

<https://www.caelum.com.br/apostila-html-css-javascript/>

PHP

<https://php.net/>

Introdução à linguagem Javascript

<http://br.ccm.net/faq/2680-javascript-introducao-a-linguagem-javascript>

REFERÊNCIAS BIBLIOGRÁFICAS

BONIATI, B., SILVA T. **Fundamentos de desenvolvimento web**. Universidade Federal de Santa Maria, Colégio Agrícola de Frederico Westphalen, 2013. ISBN: 978-85-63573-23-0.

BRITO, K. **Fundamentos do Desenvolvimento Web: Curso Técnico em Informática**. Colatina: CEAD / Ifes, 2011. ISBN: 978-85-62934-04-9. Disponível em:
edeotec.mec.gov.br/images/stories/pdf/eixo_infor.../081112_fund_de-senv.pdf.

Acessado em 23 de novembro de 2016.

COSTA, A. **Java para Desenvolvimento Web**. 2015. Disponível em:
<http://www.aeciocosta.com.br/wp-content/uploads/Curso%20Java%20Web/Apostilas/2-Java%20para%20Web%20-%20JSP.pdf>. Acessado em 13 de dezembro de 2016.

GLAZAR, J. **Programação para web: Curso técnico em informática**. Colatina: IFES, 2011. 102 p. ISBN: 978-85-62934-37-7

GONÇALVES, L. **Apostila de XHTML**. 2005. Disponível em: jkolb.com.br/wp-content/uploads/2016/06/Apostila-XHTML-Básico.pdf. Acessado em 26 de novembro de 2016.

GONÇALVES (b), L. **Apostila de JavaScript**. 2005. Disponível em: www.netsoft.inf.br/aulas/5_SIN_Programacao_Web/Apostila_javaScript.pdf. Acessado em 26 de novembro de 2016.

HARTKE NETO, R. **Curso de JSP**. Santa Catarina 2002. Disponível em: <http://pet.inf.ufsc.br/wp-content/uploads/2014/03/hartke2002jsp>.

[pdf](#). Acessado em 13 de dezembro de 2016.

MANZANO, J. A. N. G.; TOLEDO, S. A. **Guia de orientação e desenvolvimento de sites – HTML, XHTML, CSS e JavaScript/JavaScript**. São Paulo: Érica, 2008.

MARCONDES, C. A. **HTML 4.0 fundamental – A base da programação para web**. 2. ed. São Paulo: Érica, 2005.

MENGUE, F. **Curso de Java Básico**, UNICAMP 2008. Disponível em: ftp://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/java_basico.pdf. Acessado em 13 de dezembro de 2016.

NIEDERST, J. **Aprenda web design**. Rio de Janeiro: Ciência Moderna, 2002.

SILVA, M. S. **Construindo sites com CSS e (X)HTML**. São Paulo: Novatec, 2007.

TEMPLE, A., MELLO, R., CALEGARI, D., SCHIEZARO, M. **Jsp, Servlets e J2EE**. 2004. Disponível em: www.inf.ufsc.br/~bosco/downloads/livro-jsp-servlets-j2ee.pdf. Acessado em 13 de dezembro de 2016.

SOBRE O AUTOR

Vinicius Ponte Machado

CV.<http://lattes.cnpq.br/9385561556243194>



Doutor em Engenharia Elétrica e de Computação pela Universidade Federal do Rio Grande do Norte, mestre em Informática Aplicada pela Universidade de Fortaleza (2003) e graduado em Informática pela mesma instituição (1999). Atualmente é professor adjunto da Universidade Federal do Piauí e docente pesquisador do Programa de Pós-Graduação em Ciência da Computação da UFPI. Tem experiência na área de Ciência da Computação, com ênfase em Gestão do Conhecimento e Inteligência Artificial, atuando principalmente nos seguintes temas: sistemas multiagente, aprendizagem de máquina e Redes Industriais.

