

# INE5646 Programação para Web

- Tópico :

Processamento no Cliente

(estes slides fazem parte do material didático da disciplina  
INE5646 Programação para Web)

# Sumário

- Representação e Visualização de Dados
  - HTML e HTML5
  - DOM
  - CSS
- Processamento de Dados
  - JavaScript
    - Elementos da linguagem
    - Ajax
  - Bibliotecas JavaScript
    - jQuery

# Introdução

- Relembrando: sistemas distribuídos são sistemas formados por vários (pelo menos 2) programas que, tipicamente, são executados em vários (pelo menos 2) computadores.
- Aplicações para web são sistemas distribuídos com pelo menos 2 programas sendo executados em pelo menos 2 computadores (o do cliente (usuário) e o do servidor).
- **O navegador (browser) deve ser visto como uma plataforma de desenvolvimento de software** e não um simples renderizador de textos escritos em HTML.

# Introdução

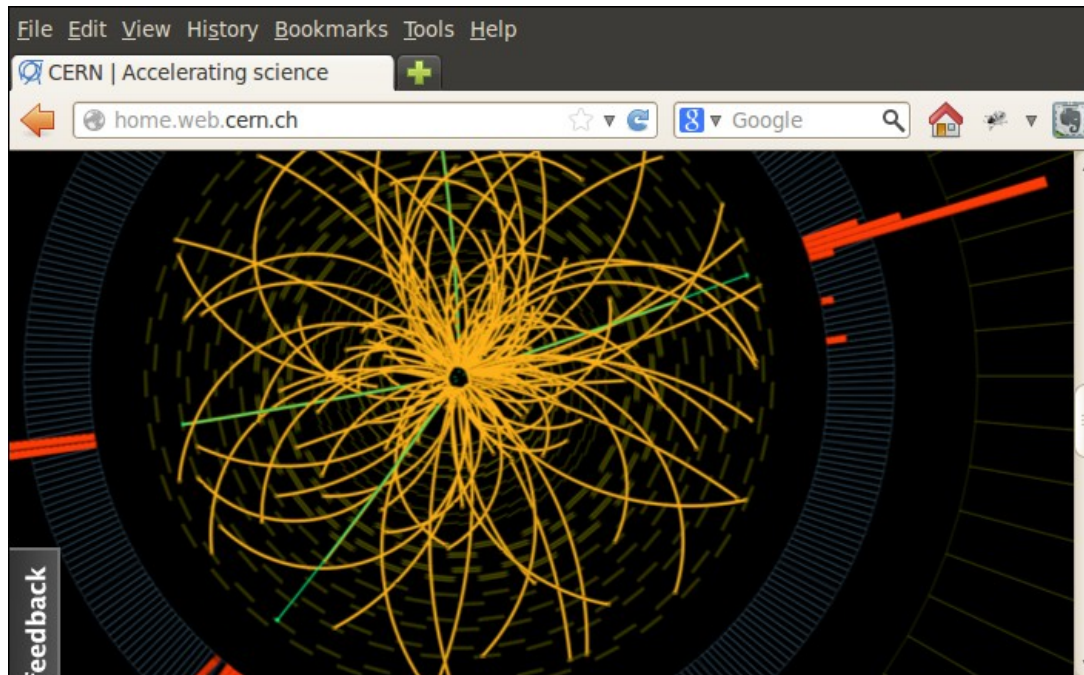
- Navegador como plataforma:
  - Representação dos dados: HTML
    - Cada tag HTML define a semântica dos dados.
    - Exemplo: `<h1>` e `<p>` são usados para representar dados semanticamente distintos.
  - Visualização dos dados: CSS
    - A aparência do dado reforça sua semântica mas é completamente desacoplada deste.
  - Processamento dos dados: JavaScript
    - A interface com o usuário da aplicação deve interagir (responder/ reagir às ações) com o usuário.
  - A especificação DOM define como estes 3 itens devem operar.

# Representação e Visualização de Dados

- **HTML** (Hyper Text Markup Language)
- **CSS** (Cascading Style Sheets)
- Analogia com processadores de texto: as folhas de estilo definem a aparência do texto mas não são o texto. Alterando as definições de uma folha de estilo altera-se, automaticamente, a aparência do texto.
- Embora cada tag HTML possua uma aparência default, o recomendado é definir a aparência via CSS.
- Assim, para cada página de texto (HTML) deve haver uma folha de estilo (CSS) associada.

# HTML e HTML5

- **HTML** (Hyper Text Markup Language)
- A linguagem foi criada pelo **físico inglês Tim Berners-Lee** no início dos anos 1990 enquanto trabalhava no **CERN** (laboratório europeu para pesquisas sobre Física).
- Seu objetivo era facilitar o compartilhamento de documentos entre os milhares de pesquisadores.



# HTML e HTML5

- Linha do tempo:
  - 1992 : HTML 1.1, ainda uma versão não padronizada.
  - 1993 : HTML 2.0
  - 1995 : HTML 3.0
  - 1996 : HTML 3.2
  - 1997 : HTML 4.0
  - 1999 : HTML 4.01
  - 2008 : HTML 5 (rascunho inicial)
- Nestes anos todos, muitas revisões, muitas indefinições levaram a uma fragmentação do mercado (cada grande “player” possui a sua versão de HTML).

# HTML e HTML5

- Ideia central: **tags** com **atributos** (opcional).
- Exemplo:
  - ``
  - `img` é a tag para referenciar uma imagem.
  - `src`, `alt`, `width`, `height` e `border` são atributos da tag `img`.
  - `imagem.gif`, `uma imagem`, `50`, `100` e `1` são os respectivos valores dos atributos.



# HTML e HTML5

- HTML tornou-se uma linguagem bastante extensa.
  - Sugestão de leitura: <http://www.w3schools.com/> apresenta toda a linguagem com exemplos interativos.
- HTML5:
  - Lançada em 2008, em meio a muitas dúvidas (conseguiria unir os principais “players”?)
  - Até hoje a especificação não está pronta.
  - Mas, ao que parece, sua adoção é inevitável.
  - Resumo da sua história:  
<http://mashable.com/2012/07/17/history-html5/>



# HTML5

- Vantagens sobre HTML 4.01:
  - Multimídia e gráficos sem depender de plugins.
  - Navegação offline.
  - Armazenamento local de dados.
  - Segurança.
  - Novas tags HTML.
  - Ampliação de CSS.
  - Comunicação mais eficiente com o servidor.
  - Acesso ao sistema de arquivos local.
- Demo: <http://slides.html5rocks.com/#landing-slide>

# HTML5

- Testes de adesão, como em <http://html5test.com/> , mostram que a adoção de HTML5 tem aumentado mas ainda há muita variação entre os navegadores.
- O uso de HTML5 neste momento ainda requer uma análise preliminar para ver quais elementos estão implementados nos principais navegadores.
- Para alguns itens, há diferenças significativas na eficiência de execução entre os diversos navegadores.
  - Exemplo: testar web workers (rotinas JavaScript que são executadas em background) em diferentes navegadores:
    - <http://www.whatwg.org/demos/workers/primes/page.html>

# Document Object Model (DOM)

- Define quais informações de uma página HTML devem ser armazenadas no navegador.
- As informações de uma página são organizadas na forma de árvore.
- Uma página HTML, no navegador, contém muitas informações além das tags HTML presentes na página.

# Document Object Model (DOM)

- Considere, como exemplo, o texto HTML abaixo:

```
exemploDOM.html x
1 <html>
2   <head>
3     <meta charset="UTF-8"/>
4     <title>INE5646 - Exemplo DOM</title>
5   </head>
6   <body>
7     <h1>INE5646 - Exemplo DOM</h1>
8     <p>Exemplo de página HTML simples</p>
9   </body>
10 </html>
```

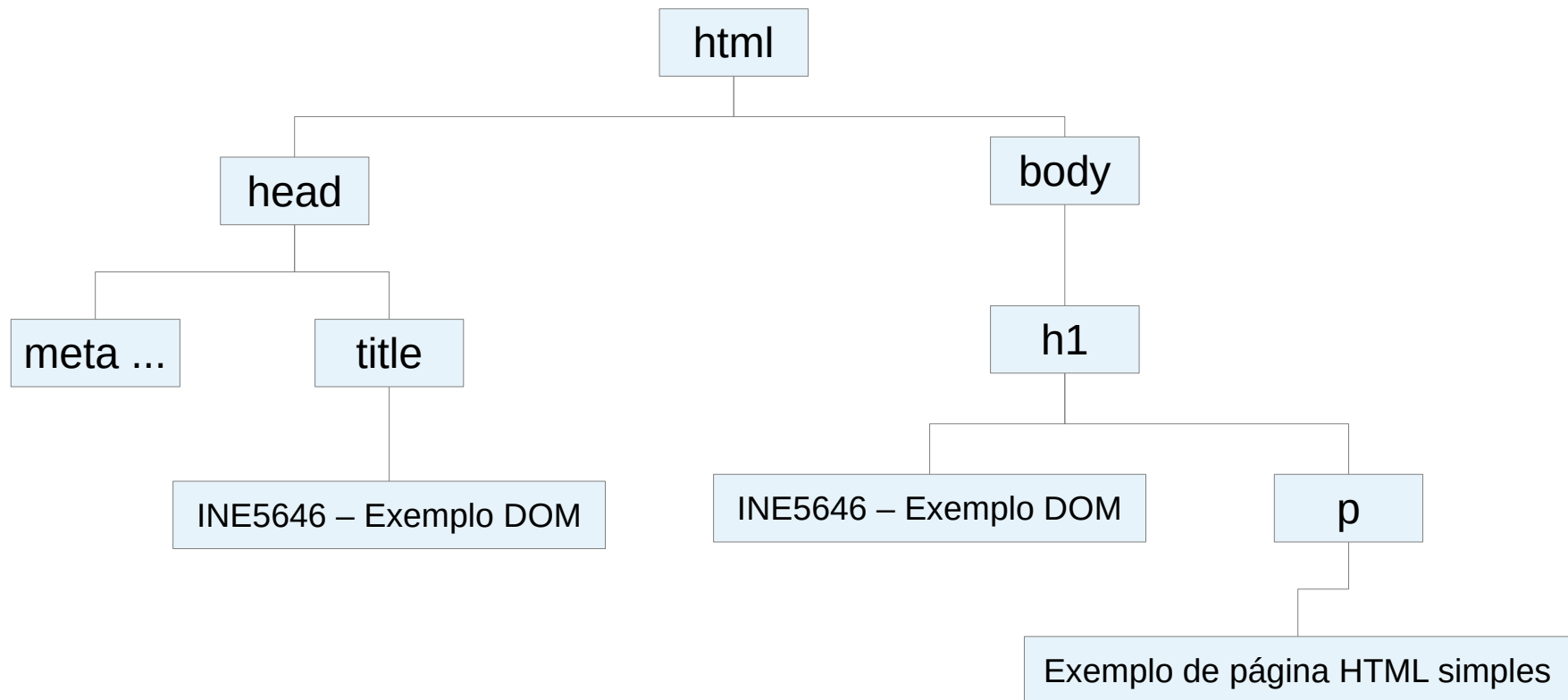
# Document Object Model (DOM)

- O texto HTML, quando renderizado pelo navegador, produz uma página com a aparência mostrada abaixo:



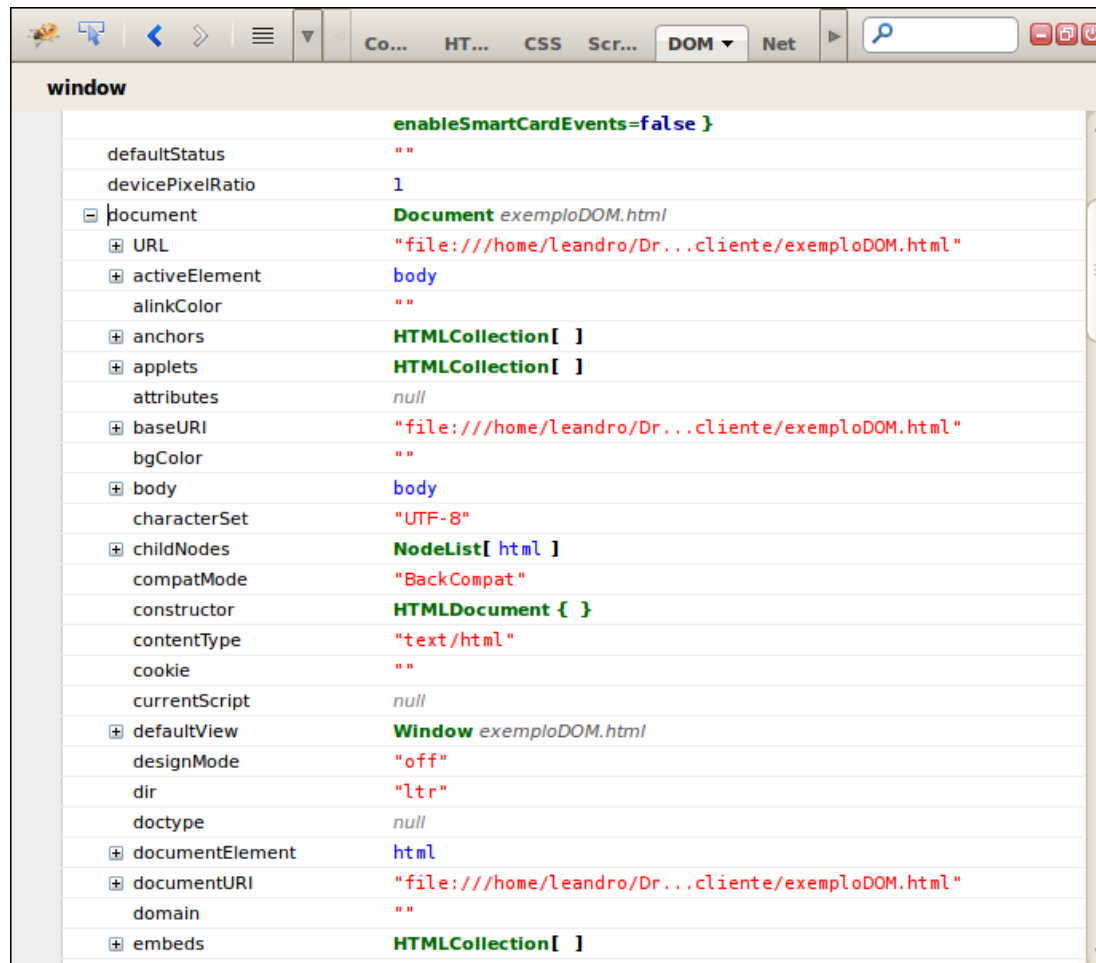
# Document Object Model (DOM)

- Toda página HTML é representada internamente no navegador como uma árvore. Na página exemplo teríamos portanto:



# Document Object Model (DOM)

- Na verdade, além das tags HTML, muitas outras informações são armazenadas (a árvore possui muitos outros nodos):





# Document Object Model (DOM)

- Cada nodo da árvore contém informações que podem ser acessadas e alteradas pelo desenvolvedor da aplicação.
- Nós podem ser adicionados/removidos.
- Os principais navegadores possuem um “editor DOM” que permite ao desenvolvedor da aplicação ter acesso à árvore DOM e realizar alterações na página que está sendo exibida.
- Qualquer alteração realizada na árvore DOM modifica, automaticamente, a aparência da página.

# Document Object Model (DOM)

- O modelo definido na especificação DOM contempla duas questões relacionadas:
  - **A representação dos dados** (tags HTML) na forma de árvore, como mostrado nas transparências anteriores.
  - A definição de **eventos associados aos dados**.
- Eventos podem ser:
  - Ações do usuário interagindo com a página.
  - Atividade da rede (ex: chegada de uma resposta HTTP).

# Document Object Model (DOM)

- **O conceito de evento define um modelo de programação:**
  - O browser se encarrega de:
    - Monitorar a ação do usuário (ou da rede).
    - Obter os dados relativos à ação (como o tipo da ação (clique, duplo clique, passagem do mouse sobre uma tag, etc), qual o nodo envolvido, etc).
    - Fazer com que os “*listeners*” (funções JavaScript) associados ao nodo sejam executados.
    - Propagar (*bubbling*) o evento em direção à raiz da árvore, permitindo que todos os listeners no caminho sejam executados.
  - **O desenvolvedor da aplicação deve criar os *listeners* e vinculá-los aos nodos da árvore.**

# Document Object Model (DOM)

- Especificações do W3C:
  - <http://www.w3.org/TR/DOM-Level-3-Core/> de 2004 para a estrutura de árvore.
  - <http://www.w3.org/TR/DOM-Level-3-Events/> de 2012 para os eventos.
- Especificações do WHATWG:
  - <http://dom.spec.whatwg.org/>
  - O WHATWG (<http://www.whatwg.org/>) é uma dissidência do W3C e foi criado em 2004.
- Embora as ideias do DOM sejam amplamente aceitas (representação da página em forma de árvore e o conceito de eventos), cada navegador possui uma implementação, em maior ou menor grau, compatível com a especificação oficial.
- Para remediar esta situação, o mercado começou a criar bibliotecas JavaScript cross-browser como, por exemplo, a jQuery.

# Cascading Style Sheets (CSS)

- Um arquivo CSS (.css) define a aparência (tipo de fonte, tamanho e cor) e posição das tags HTML presentes em uma página.
- O conceito fundamental chama-se **seletor**:
  - Um seletor é uma expressão textual que determina quais nodos serão afetados (terão sua aparência default alterada) pelas definições contidas no seletor.
- Embora os seletores possam ser definidos na própria página HTML, o recomendado é que o sejam em um arquivo próprio pois simplifica sua manutenção e possibilita a sua reutilização em outras páginas.

# CSS - Exemplo

- O texto HTML abaixo não usa CSS. Portanto a localização e a aparência das tags dentro de `<body>` são as default.

```
index_sem_css.html x
1  <html>
2    <head>
3      <meta charset="UTF-8"/>
4      <title>INE5646 - Exemplo 1 CSS</title>
5    </head>
6    <body>
7      <h1>INE5646 - Exemplo 1 CSS</h1>
8      <p>Esta página não usa CSS.</p>
9      <p>Cada tag usa sua aparência default.</p>
10   </body>
11 </html>
```



# CSS - Exemplo

- O texto HTML abaixo usa CSS.
- A linha 5 indica que o navegador deve obter o arquivo estilos.css e aplicar os seletores para alterar a aparência do texto HTML.

```
index_com_css.html x
1 <html>
2   <head>
3     <meta charset="UTF-8"/>
4     <title>INE5646 - Exemplo 1 CSS</title>
5     <link rel="stylesheet" type="text/css" href="estilos.css">
6   </head>
7   <body>
8     <h1>INE5646 - Exemplo 1 CSS</h1>
9     <p>Esta página usa CSS.</p>
10    <p>A aparência é definida em estilos.css</p>
11  </body>
12 </html>
```

# CSS - Exemplo

- O arquivo estilos.css define 3 seletores:
  - Linhas 1 a 3: **seletor body**: todos os nodos contidos na subárvore onde body é a raiz devem ser escritos em cor vermelha.
  - Linhas 5 a 7: **seletor h1**: todos os nodos da página vinculados à tag h1 devem ter fundo azul.
  - Linhas 9 a 11: **seletor p**: todos os nodos da página vinculados à tag p devem ser escritos em verde e o tamanho da fonte deve ser 50% maior que o tamanho default.

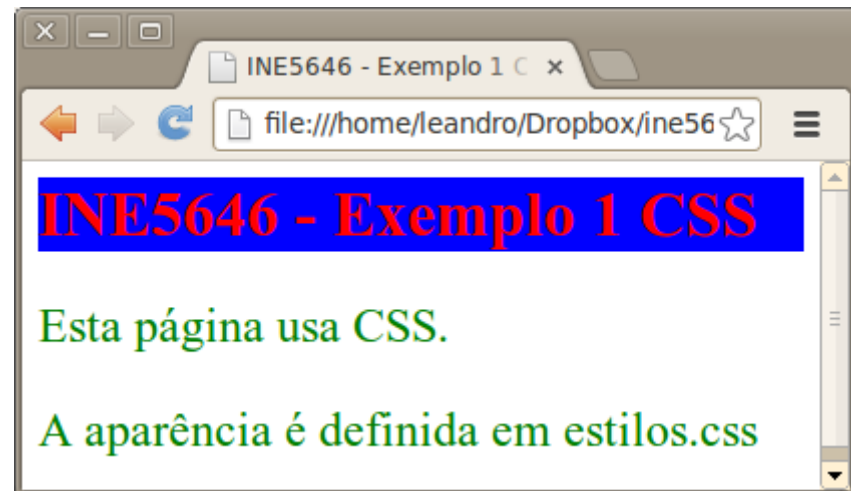
```
estilos.css
1  body {
2      color: red;
3  }
4
5  h1 {
6      background-color: blue;
7  }
8
9  p {
10     font-size: 1.5em;
11     color: green;
12 }
```



# CSS - Exemplo

- Na figura abaixo, observa-se que:
  - Os seletores obedecem à estrutura de árvore e as definições em um nodo aplicam-se a todas as subárvores daquele nodo. É por isso que o texto de h1 está escrito em vermelho (herdado de body).
  - Os seletores aplicam-se a todos os nodos que casam com a expressão que define o seletor. É por isso que os dois nodos p aparecem em verde e em um tamanho 50% maior do que o normal.

```
estilos.css
1  body {
2      color: red;
3  }
4
5  h1 {
6      background-color: blue;
7  }
8
9  p {
10     font-size: 1.5em;
11     color: green;
12 }
```



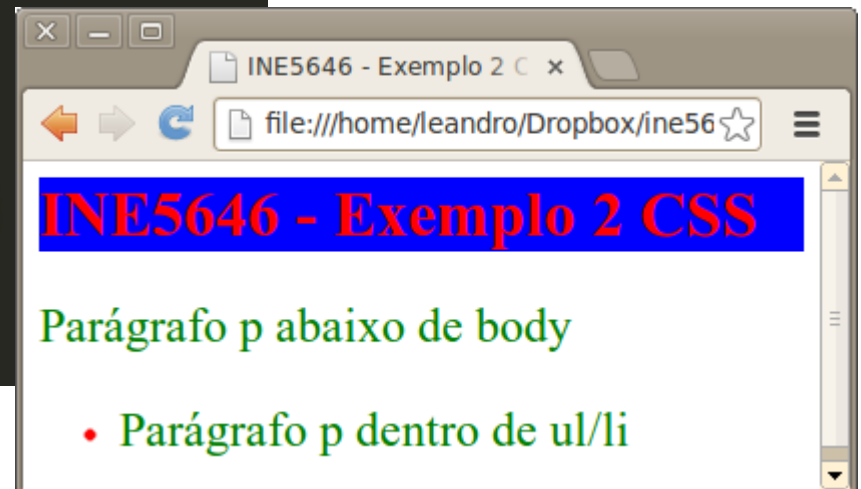
# Seletores CSS

- No exemplo mostrado, todos os 3 seletores eram formados por expressões que coincidiam com as tags HTML (body, h1 e p).
- Porém, as expressões podem ser mais sofisticadas, permitindo filtrar, com alto grau de refinamento, os nodos desejados.
- **Atenção:** a aplicação dos seletores tem um custo computacional inerente ao processo de seleção das subárvores. Por exemplo: o seletor p usado no exemplo significa, em termos genéricos o seguinte:
  - Dada uma árvore HTML, retorne todas as subárvores cujo nodo raiz tenha a tag p.

# Seletores CSS

- Por exemplo: o seletor “p” inclui o parágrafo que está dentro da árvore “ul”. Isso mostra que a coleta percorre todos os nodos da árvore (consumindo tempo de processamento).

```
index2_com_css.html x
1 <html>
2   <head>
3     <meta charset="UTF-8"/>
4     <title>INE5646 - Exemplo 2 CSS</title>
5     <link rel="stylesheet" type="text/css" href="estilos.css">
6   </head>
7   <body>
8     <h1>INE5646 - Exemplo 2 CSS</h1>
9     <p>Parágrafo p abaixo de body</p>
10    <ul>
11      <li><p>Parágrafo p dentro de ul/li</p></li>
12    </ul>
13  </body>
14 </html>
```



# Seletores CSS

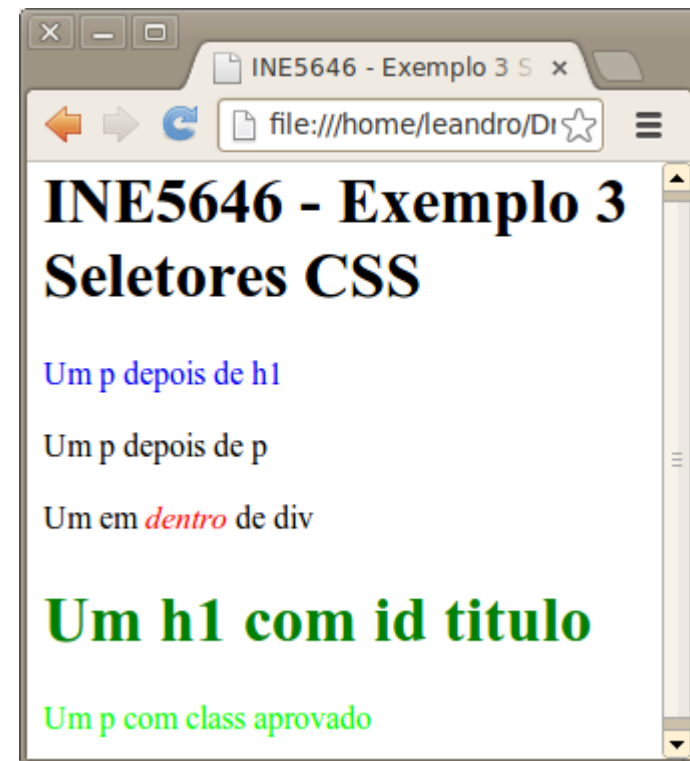
- Para alguns exemplos de seletores (mostrados a seguir), considere a página HTML abaixo:

```
index_seletores.html x
1  <html>
2    <head>
3      <meta charset="UTF-8"/>
4      <title>INE5646 - Exemplo 3 Seletores CSS</title>
5      <link rel="stylesheet" type="text/css" href="seletores.css">
6    </head>
7    <body>
8      <h1>INE5646 - Exemplo 3 Seletores CSS</h1>
9      <p>Um p depois de h1</p>
10     <p>Um p depois de p</p>
11     <div>
12       <p>Um em <em>dentro</em> de div</p>
13     </div>
14     <h1 id="titulo">Um h1 com id titulo</h1>
15     <p class="aprovado">Um p com class aprovado
16   </body>
17 </html>
```

# Seletores CSS

- Alguns exemplos de seletor:

```
seletores.css
1  /* exemplos de seletores */
2  h1+p {
3      color: blue;
4  }
5
6  div em {
7      color: red;
8  }
9
10 #titulo {
11     color: green;
12 }
13
14 .aprovado {
15     color: lime;
16 }
```



# Seletores CSS

- Outros exemplos de seletores:
  - DIV,P : seleciona todos as DIV E todos os P.
  - DIV P : seleciona todos os P que estejam dentro de uma DIV.
  - DIV>P : seleciona todos os P que tenham DIV como nodo pai.
  - DIV+P : seleciona todos os P que aparecem depois de uma DIV.
  - A[src\$=".pdf"] : seleciona todos os A cujo atributo src termine com a substring “.pdf”.
  - A[src\*="ufsc"] : seleciona todos os A cujo atributo src contenha a substring “ufsc”.

# Propriedades CSS

- As propriedades CSS definem quais alterações podem ser feitas nos nodos selecionados pelos seletores.
- Tipicamente altera-se:
  - A cor (color, background-color, etc)
  - A fonte (font-size, font-family, font-style, etc)
  - O tamanho (height, width, max-height, etc)
  - A posição em relação a outros elementos (position, float)
- Para cada categoria, há muitas propriedades.
- Ver em <http://www.w3schools.com/css>

# CSS – Box Model

- Cada nodo da árvore, isto é, cada tag do documento HTML, é representado visualmente por um retângulo.



Fonte: [http://www.w3schools.com/css/css\\_boxmodel.asp](http://www.w3schools.com/css/css_boxmodel.asp)



# CSS – Box Model

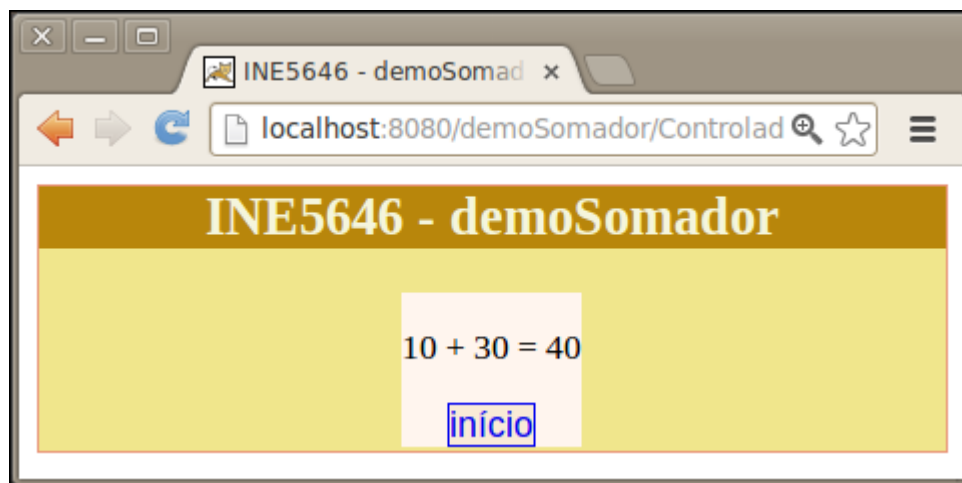
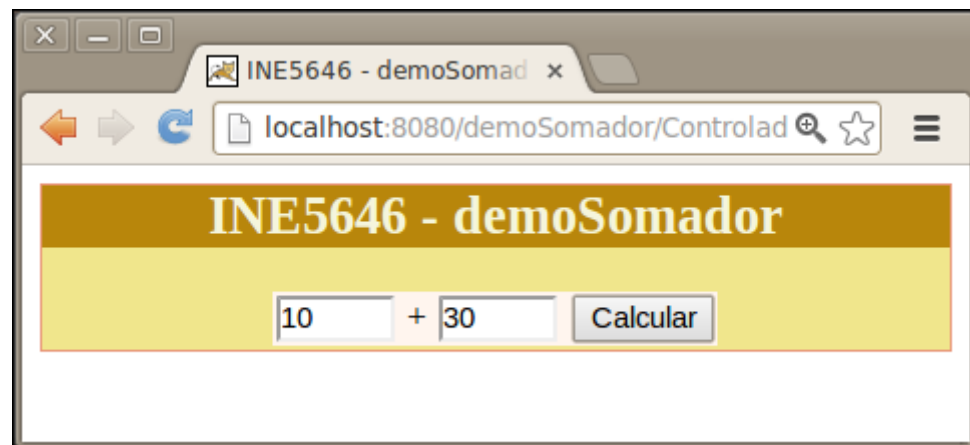
- Uma página é, portanto, formada por uma sequência de linhas que contêm blocos.
- Uma linha pode:
  - Conter um único bloco. Exemplo: tag H1.
  - Conter vários blocos. Exemplo: tags SPAN e A.
  - A propriedade ***display*** permite definir que:
    - **display: block** :: na linha, o bloco é único e ocupa todo o espaço disponível.
    - **display: inline** :: na linha, o bloco fica ao lado do bloco anterior e ocupa o espaço proporcional ao seu conteúdo.
    - **display: inline-block** :: na linha, o bloco é único e ocupa o espaço proporcional ao seu tamanho.
    - **display: none** :: na linha, o bloco não ocupa nenhum espaço.

# CSS – Aplicação Exemplo

- A aplicação exemplo:
  - Permite que o usuário some dois números inteiros.
  - Todo processamento é feito no servidor (web 1.0).
  - Utiliza a tecnologia JSP.
  - Possui 3 páginas:
    - inicial.jsp : página inicial.
    - soma.jsp : obter os números a serem somados.
    - resultado.jsp : mostra o resultado da soma.
  - Possui 2 fragmentos de código JSP:
    - \_cabecalho.jspf : cabeçalho das páginas.
    - \_msg.jspf : área para mensagens de erro.

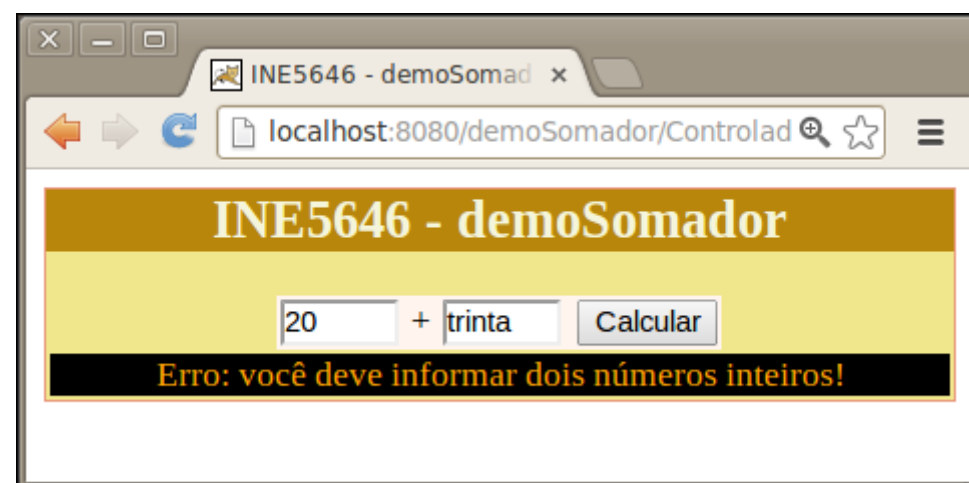
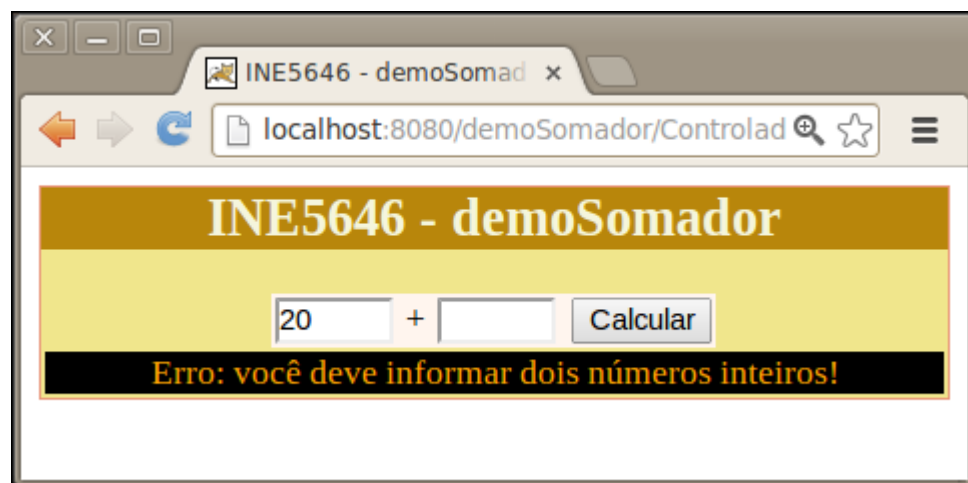
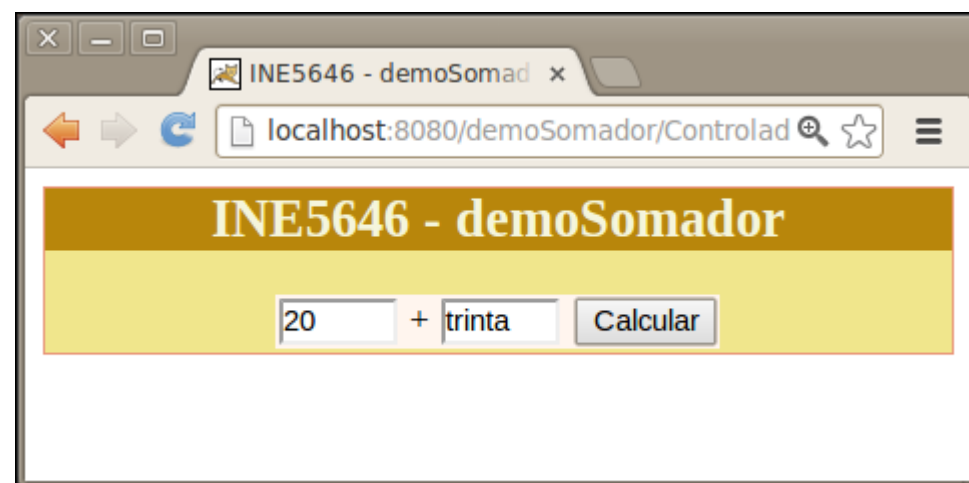
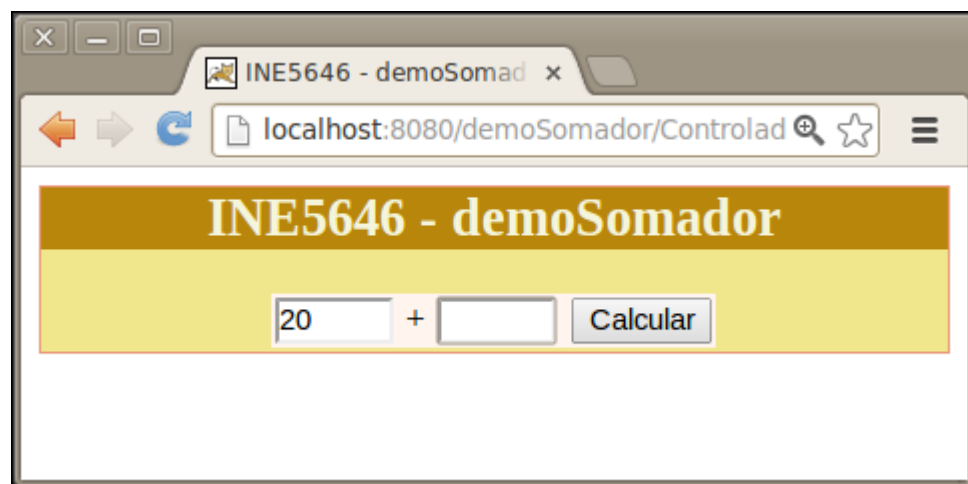
# CSS – Aplicação Exemplo

- Páginas:



# CSS – Aplicação Exemplo

- Os dois campos devem ser preenchidos corretamente.



# inicial.jsp

- Linha 7 : define os estilos CSS usados na página.
- Note o uso da tag **div** para estruturar o conteúdo da página.

```
inicial.jsp x
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>INE5646 - demoSomador :: inicial</title>
7     <link rel="stylesheet" type="text/css" href="css/estilos.css"/>
8   </head>
9   <body>
10    <div id="app">
11      <%@include file="WEB-INF/jspf/_cabecalho.jspf" %>
12      <div id="conteudo">
13        <a href="Controlador?acao=2">entrar</a>
14      </div>
15      <%@include file="WEB-INF/jspf/_msg.jspf" %>
16    </div>
17  </body>
18 </html>
```

# soma.jsp

- Linha 14: note que a requisição será enviada ao controlador (MVC).

```
soma.jsp x
1  <%@page contentType="text/html" pageEncoding="UTF-8"%>
2  <!DOCTYPE html>
3  <html>
4  <head>
5      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6      <title>INE5646 - demoSomador :: soma</title>
7      <link rel="stylesheet" type="text/css" href="css/estilos.css"/>
8  </head>
9  <body>
10     <div id="app">
11         <%@include file="WEB-INF/jspf/_cabecalho.jspf" %>
12         <div id="conteudo">
13             <div class="painel">
14                 <form action="Controlador?acao=3" method="post">
15                     <input type="text" size="5" name="n1" value="${n1}"/> +
16                     <input type="text" size="5" name="n2" value="${n2}"/>
17
18                     <input type="submit" value="Calcular"/>
19                 </form>
20             </div>
21         </div>
22         <%@include file="WEB-INF/jspf/_msg.jspf" %>
23     </div>
24 </body>
25 </html>
```



# resultado.jsp

- Linha 14: note que a navegação entre páginas é administrada pelo controlador.

```
resultado.jsp x
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>INE5646 - demoSomador :: resultado</title>
7     <link rel="stylesheet" type="text/css" href="css/estilos.css"/>
8   </head>
9   <body>
10    <div id="app">
11      <%@include file="WEB-INF/jspf/_cabecalho.jspf" %>
12      <div id="conteudo">
13        <p>${resposta}</p>
14        <a href="Controlador?acao=1">início</a>
15      </div>
16      <%@include file="WEB-INF/jspf/_msg.jspf" %>
17    </div>
18  </body>
19 </html>
```

# \_cabecalho.jspf e \_msg.jspf

- Note o uso da tag `div` para caracterizar o papel que estes fragmentos de código JSP realizam dentro das páginas.

```
_cabecalho.jspf x
1 <%@ page pageEncoding="UTF-8" %>
2 <div id="cabecalho">
3   <h2>INE5646 - demoSomador</h2>
4 </div>
```

```
_msg.jspf x
1 <%@ page pageEncoding="UTF-8" %>
2 <div id="msg">
3   ${msg}
4 </div>
```



# estilos.css

```
estilos.css
1  #app {
2    border: darksalmon solid thin;
3    background-color: khaki;
4    text-align: center;
5  }
6
7  #cabecalho {
8    background-color: darkgoldenrod;
9  }
10
11 #cabecalho h2 {
12   margin-top: 0px;
13   color: beige;
14 }
15
16 #conteudo {
17   background-color: seashell;
18   display: inline-block;
19 }
20
21 #msg {
22   background-color: black;
23   color: orange;
24   margin: 2px;
25 }
26
27 a {
28   background-color: antiquewhite;
29   text-decoration: none;
30   font-family: sans-serif;
31   border: solid 1px;
32 }
```

- **#app** : todo conteúdo da aplicação estará contido dentro de um bloco:
  - Com borda cor darksalmon, linha sólida e fina.
  - Com cor de fundo khaki.
  - Com textos centralizados.

# estilos.css

```
estilos.css
1  #app {
2    border: darksalmon solid thin;
3    background-color: khaki;
4    text-align: center;
5  }
6
7  #cabecalho {
8    background-color: darkgoldenrod;
9  }
10
11 #cabecalho h2 {
12   margin-top: 0px;
13   color: beige;
14 }
15
16 #conteudo {
17   background-color: seashell;
18   display: inline-block;
19 }
20
21 #msg {
22   background-color: black;
23   color: orange;
24   margin: 2px;
25 }
26
27 a {
28   background-color: antiquewhite;
29   text-decoration: none;
30   font-family: sans-serif;
31   border: solid 1px;
32 }
```

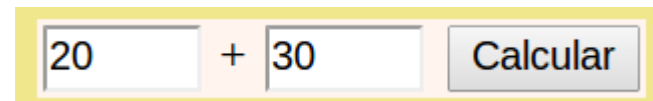
- **#cabecalho** : O cabeçalho das páginas terá:
  - Cor de fundo darkgoldenrod.
- **#cabecalho h2** : O título do cabeçalho terá:
  - Margem 0 na parte superior.
  - Cor bege.

INE5646 - demoSomador

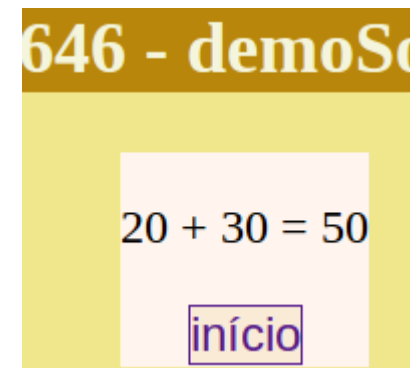
# estilos.css

```
estilos.css
1  #app {
2    border: darksalmon solid thin;
3    background-color: khaki;
4    text-align: center;
5  }
6
7  #cabecalho {
8    background-color: darkgoldenrod;
9  }
10
11 #cabecalho h2 {
12   margin-top: 0px;
13   color: beige;
14 }
15
16 #conteudo {
17   background-color: seashell;
18   display: inline-block;
19 }
20
21 #msg {
22   background-color: black;
23   color: orange;
24   margin: 2px;
25 }
26
27 a {
28   background-color: antiquewhite;
29   text-decoration: none;
30   font-family: sans-serif;
31   border: solid 1px;
32 }
```

- **#conteudo** : O conteúdo das páginas terá:
  - Cor de fundo seashell.
  - Terá o tamanho proporcional às informações exibidas.



20 + 30 Calcular



# estilos.css

```
estilos.css
1  #app {
2    border: darksalmon solid thin;
3    background-color: khaki;
4    text-align: center;
5  }
6
7  #cabecalho {
8    background-color: darkgoldenrod;
9  }
10
11 #cabecalho h2 {
12   margin-top: 0px;
13   color: beige;
14 }
15
16 #conteudo {
17   background-color: seashell;
18   display: inline-block;
19 }
20
21 #msg {
22   background-color: black;
23   color: orange;
24   margin: 2px;
25 }
26
27 a {
28   background-color: antiquewhite;
29   text-decoration: none;
30   font-family: sans-serif;
31   border: solid 1px;
32 }
```

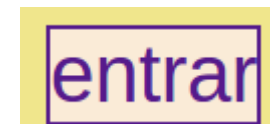
- **#msg** : A área de mensagem de erro:
  - Cor de fundo black.
  - Cor do texto orange.
  - Margem de 2 pixels.

Erro: você deve informar dois números inteiros!

# estilos.css

```
estilos.css
1  #app {
2    border: darksalmon solid thin;
3    background-color: khaki;
4    text-align: center;
5  }
6
7  #cabecalho {
8    background-color: darkgoldenrod;
9  }
10
11 #cabecalho h2 {
12   margin-top: 0px;
13   color: beige;
14 }
15
16 #conteudo {
17   background-color: seashell;
18   display: inline-block;
19 }
20
21 #msg {
22   background-color: black;
23   color: orange;
24   margin: 2px;
25 }
26
27 a {
28   background-color: antiquewhite;
29   text-decoration: none;
30   font-family: sans-serif;
31   border: solid 1px;
32 }
```

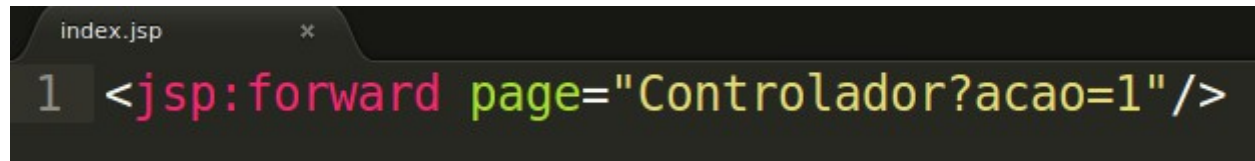
- **a** : Os links das páginas terão uma aparência completamente diferente da default:
  - Cor de fundo antiquewhite.
  - Texto não sublinhado (linha 29).
  - Fonte sans-serif
  - Borda sólida de 1 pixel.





# Aplicação Exemplo: Controlador

- A especificação servlet define que, por default, a página inicial de uma aplicação JSP se chame index.jsp.
- Assim, a requisição **http://localhost:8080/demoSomador** é encaminhada para a página (servlet) index.jsp que, por sua vez, encaminha para o controlador.



```
index.jsp x
1 <jsp:forward page="Controlador?acao=1"/>
```

# Controlador.java

- A classe Controlador representa o controlador (MVC) da aplicação.
- Linha 10: Toda requisição HTTP que **iniciar** com `http://localhost:8080/demoSomador/Controlador` será tratada pelo controlador.

```
Controlador.java x
1  package ine5646.demosomador.controlador;
2
3  import java.io.IOException;
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet(name = "Controlador", urlPatterns = {"/Controlador/*"})
11 public class Controlador extends HttpServlet {
```

# Controlador.java

- Obtém a ação desejada (linha 26) e define qual página será enviada como resposta (variável destino).

```
23     protected void processRequest(HttpServletRequest request,
24                                   HttpServletResponse response)
25         throws ServletException, IOException {
26         String acao = request.getParameter("acao");
27         String destino = "";
28         if (acao != null) {
29             switch (acao) {
30                 case "1":
31                     destino = "inicial.jsp";
32                     break;
33                 case "2":
34                     destino = "soma.jsp";
35                     break;
36                 case "3":
37                     destino = analisePedidoDeSoma(request);
38                     break;
39             }
40         }
41         if (destino.equals("")) {
42             request.setAttribute("msg", "Ação inválida!");
43             destino = "index.jsp";
44         }
45         request.getRequestDispatcher(destino).forward(request, response);
46     }
```



# Controlador.java

- Para um pedido de soma, se os parâmetros “n1” e “n2” forem válidos, retornar a página “resultado.jsp”. Caso contrário, retornar a página “soma.jsp” indicando o erro.
- Em uma aplicação mais complexa o mais aconselhado seria extrair os dados da requisição e passá-los para um objeto do domínio do problema (model – MVC).

```
89 private String analisePedidoDeSoma(HttpServletRequest request) {
90     String destino = "resultado.jsp";
91     String sN1 = request.getParameter("n1");
92     String sN2 = request.getParameter("n2");
93     try {
94         int n1 = Integer.parseInt(sN1);
95         int n2 = Integer.parseInt(sN2);
96         int soma = n1 + n2;
97         request.setAttribute("resposta", "" + n1 + " + " + n2 + " = " + soma);
98
99     } catch (NumberFormatException ex) {
100         request.setAttribute("msg", "Erro: você deve informar dois números inteiros!");
101         request.setAttribute("n1", sN1);
102         request.setAttribute("n2", sN2);
103
104         destino = "soma.jsp";
105     }
106     return destino;
107 }
```

# Processamento de Dados

- Todos os conceitos vistos até agora (HTML, HTML5, DOM e CSS) representam elementos estáticos da aplicação (camada 1).
- A dinâmica de interatividade da camada 1 acontece programaticamente por meio de código escrito em **JavaScript**.
- JavaScript, no contexto do desenvolvimento de aplicações para web, deve ser percebida como uma linguagem de programação completa e não uma simples linguagem para produzir animações em páginas HTML.

# JavaScript – Elementos da Linguagem

- **Comentários:**  
`//` comentário em única linha  
`/*` comentário em  
    mais de uma linha  
`*/`
- **Palavras reservadas:**  
break, case, catch, class, const, continue, debugger, default,  
delete, do, else, enum, export, extends, false, finally, for,  
function, if, implements, import, in, instanceof, interface, let,  
new, null, package, private, protected, public, return, static,  
super, switch, this, throw, true, try, typeof, var, void, while,  
with, yield.
- Baseada no padrão ECMAScript - <http://www.ecmascript.org/>

# JavaScript – Variáveis e Operadores

- Um programa é formado por uma sequência de declarações (statements) separados por “;”
- **Variáveis:**  
**var v1** = 10;  
**var v2** = “testando”;
- **Operadores aritméticos:** +, -, \*, / , % (resto da divisão), ++ e --.
- **Operadores de atribuição:** =, +=, -=, \*=, /= e %=  
Exemplo: x += y equivale à x = x+ y
- **Operadores lógicos:** &&, ||, !, ==, !=, >, >=, <, <=, === (mesmo valor e tipo), !==,

# JavaScript – seleção e repetição

- **Seleção**: if, if-else, switch (como em Java)
- **Repetição**: while, do-while, for, for-in

Exemplo:

```
var pessoa = {nome: "Fulano",  
              sobrenome: "de Tal"};
```

```
for (atributo in pessoa)  
  pessoa[atributo];
```

- Neste comando for, a variável atributo assumirá os valores “nome” e “sobrenome”.

# JavaScript – objetos (classes)

- **Object** – para criar objetos  
(exemplo será mostrado mais adiante)
- **Number** – números decimais 64 bits  
`var n = new Number(55.28);`  
`var p = 55.28;`
- **Math** – funções matemáticas (as tradicionais)  
`var raiz = Math.sqrt(16);`
- **Date** – data  
`var d = new Date();`  
`d.getDay(); // retorna entre 0 e 6`  
`d.getMonth(); // retorna entre 0 e 11`

# JavaScript – objetos (classes)

- **String** – para criar strings  
com uma grande variedade de métodos  
`var s = “Fulano de Tal”;`  
`s.toLowerCase(); // “fulano de tal”`
- **RegExp** – expressão regular  
`var er = new RegExp(“[a-z]+”);`  
`var er = /[a-z]+; // pelo menos 1 letra`  
`er.test(“teste”); // true`
- **Boolean** – true ou false

# JavaScript – Arrays

- Conjunto de dados.  
var v1 = **new Array**("bem", "vindo");  
var v2 = **[**"bem", "vindo"**]**;  
var v3 = **[]**; // array vazio  
v3[0] = "bem"; // array aumenta se necessário
- Propriedade **length** : tamanho do array.  
v2.**length** // retorna 2
- Método **concat**: une dois arrays  
var v1 = [1,2,3];  
var v2 = [4,5,6];  
var v3 = v1.concat(v2); // [1,2,3,4,5,6]



# JavaScript – Arrays

- Método **join**: retorna string com dados do array  
var v1 = [1,2,3];  
var s1 = v1.join();// retorna “1,2,3”  
var s2 = v1.join(“+”); // retorna “1+2+3”
- Método **push**: adiciona um dado no final do array  
var v1 = [];  
v1.push(1); // [1]  
v1.push(2); // [1,2]
- Método **pop**: remove o último dado do array

# JavaScript – Arrays

- Método **reverse**: inverte a ordem dos dados  
`var v1 = [1,2,3];`  
`v1.reverse(); // [3,2,1]`
- Método **shift**: remove o primeiro dado  
`var v1 = [1,2,3];`  
`v1.shift(); // [2,3]`
- Método **unshift**: adiciona o primeiro dado  
`var v1 = [2,3];`  
`v1.unshift(1); // [1,2,3]`
- Método **slice**: extrai uma parte do array  
`var v1 = [1,2,3,4,5];`  
`var v2 = v1.slice(3); // [4,5]`

# JavaScript – Arrays

- Método **splice**: substitui uma parte dos dados por outros dados

```
var v1 = [1,2,3,4,5,6];
```

```
v1.splice(1,2,10,20,30); // [1,10,20,30,4,5,6]
```

1 – índice do início da remoção

2 – quantos dados serão removidos

10,20,30,... - dados que serão inseridos a partir do índice

- Método **sort**: ordena os dados

```
var v1 = [8,3,6,2];
```

```
v1.sort(); // [2,3,6,8]
```

# JavaScript – Arrays

- Método **forEach**: percorre o array aplicando uma função em cada dado do array. A função pode ter até 3 parâmetros:
  - O primeiro é o dado;
  - O segundo é o índice do dado;
  - O terceiro é o próprio array;
- Exemplo:

```
function dobre(e, i, a) {a[i] = e * 2;}  
var v1 = [1,2,3];  
v1.forEach(dobre); // [2,4,6]
```

# JavaScript – Funções

- **Funções são valores**. Bem vindo à programação funcional!!!
  - Podem ser atribuídas à variáveis.
  - Podem ser passadas como parâmetro de uma função.
  - Podem ser retornadas como resultado da aplicação de uma função.
- Função tradicional:  
`function dobre(n) {return n *2;}`  
`var v = dobre(5); // atribuirá o valor 10 à variável v.`
- Como valor de variável:  
`var d = dobre;`  
`d(10); // retornará 20`  
`var triplique = function (n) {return n * 3;}`  
`triplique(10); // retornará 30`

# JavaScript – Funções

- **Como parâmetro:**

```
function mais5(fn, n) {return fn(n) + 5;}  
var v = mais5(dobre, 10); // retornará 25
```

- **Como valor resultante de uma função:**

```
function dobre(n) {return n * 2;}  
function triplique(n) {return n * 3;}  
function determineFator(v, f1, f2) {  
    if (v < 16)  
        return f1;  
    else  
        return f2;  
}  
var fator = determineFator(30, dobre, triplique);  
fator(100); // retornará 300
```

- Obs: a função determineFator também poderia retornar uma função anônima:  
return function (n) {....};

# JavaScript – Objetos

- Um **objeto** contém um ou mais pares chave-valor.

- A chave pode ser qualquer identificador JavaScript;
- O valor pode ser qualquer valor JavaScript:
  - Número, string, array, função, objeto

- Exemplo:

```
var pessoa = {  
  nascimento: 1966,  
  nome: "Fulano de Tal",  
  maisVelhoQue: function(ano) {  
    return nascimento > ano;  
  };  
};
```

```
pessoa.nascimento; // 1966
```

```
pessoa["nome"]; // "Fulano de Tal" - note esta outra forma de acesso
```

```
pessoa.maisVelhoQue(1999); // false
```

# JavaScript e DOM

- Todas as informações contidas em uma página (representação DOM) são acessáveis (leitura/modificação) por meio do objeto **document**.
  - Exemplos:
    - document.title : o título da página
    - document.links : retorna um array com os links contidos na página
    - document.forms : retorna um array com os forms contidos na página
- Cada tag HTML é representada por um objeto específico: Button, Link, Image, Form, etc.



# JavaScript e DOM

- **O grande problema:** incompatibilidade entre os navegadores:
  - Na árvore DOM, alguns nomes de classes, atributos e métodos dos objetos que representam as tags de página HTML são diferentes ou nem mesmo estão implementados.
  - O pior dos pesadelos para o desenvolvedor de aplicações:
    - Ter que desenvolver uma versão da aplicação para cada navegador.
    - Ter que testar cada uma das versões.
- **A solução:** bibliotecas cross-browser que escondem do desenvolvedor as incompatibilidades.
  - Exemplo mais famoso: jQuery - <http://jquery.com/>

# JavaScript e Ajax

- **Ajax**: Asynchronous JavaScript and XML
- Sem ajax:
  - Toda requisição HTTP ao servidor significa: “prezado servidor, queira por favor retornar a página HTML identificada pela URL indicada nesta requisição”.
  - Enquanto a resposta não chega, o browser (interface com o usuário da aplicação) fica congelada pois a requisição é síncrona.
- Com ajax:
  - A requisição tem o objetivo de solicitar dados ou fragmentos de página que serão incorporados na página atual.
  - A requisição é assíncrona, ou seja, o browser continua respondendo às ações do usuário.
  - Quando a resposta HTTP chega, uma função JavaScript é executada para atualizar a página que está sendo exibida.

# JavaScript – em página HTML

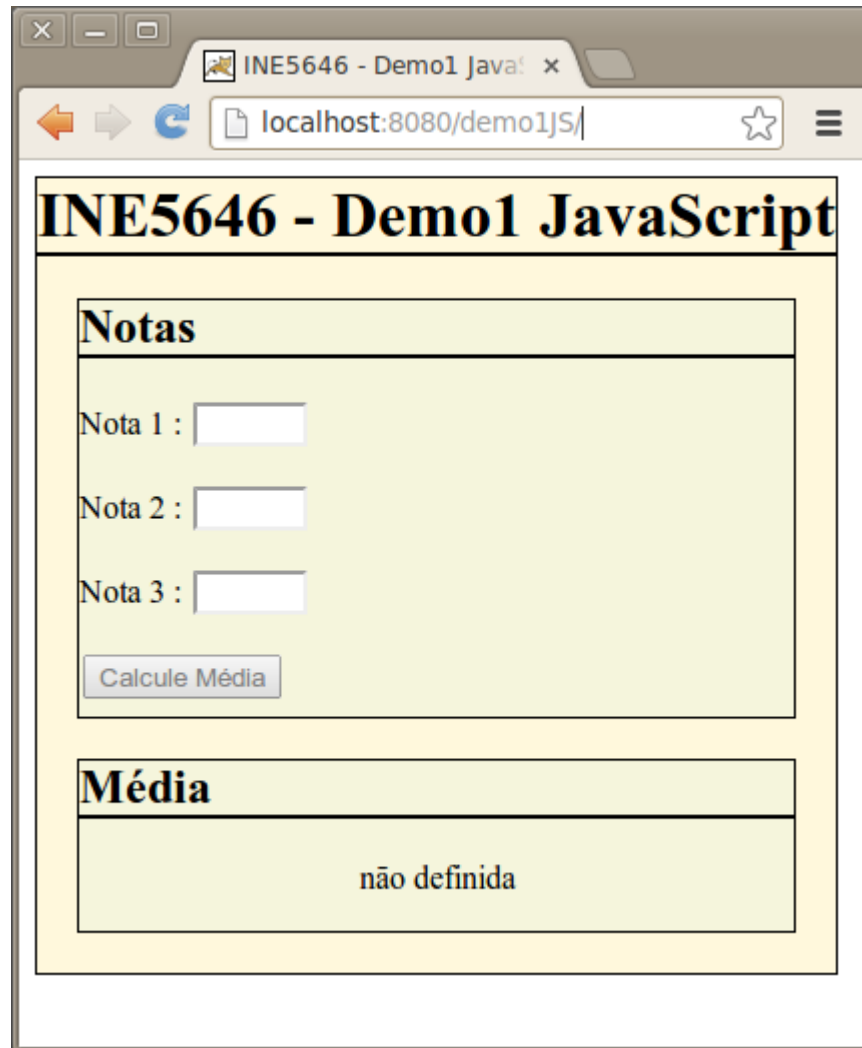
- Três formas para incorporar código na página HTML:
  - Arquivo externo (extensão .js). Exemplo:  
`<script src="arquivo.js"/>`
  - Na própria página. Exemplo:  
`<script>`  
    código JavaScript  
`</script>`
  - Como valor de atributo de tag HTML (não recomendado).
    - Exemplo:  
`<button onClick="código JavaScript">Ok</button>`

# JavaScript – Aplicação Exemplo

- A aplicação exemplo permite que o usuário calcule a média de suas três notas. As notas devem estar no intervalo  $[0, 10]$ .
- Se a média for maior ou igual a 5.75 então o aluno está aprovado.
- A aplicação contém uma única página e todo o processamento (validação dos dados digitados e atualização da página) é realizado na camada 1 (browser) via JavaScript. Não há comunicação com a camada 2.

# Aplicação Exemplo – index.jsp

- Página em seu estado inicial.



The screenshot shows a web browser window with the title "INE5646 - Demo1 JavaScript". The address bar displays "localhost:8080/demo1JS/". The main content area is divided into two sections:

**Notas**

Nota 1 :

Nota 2 :

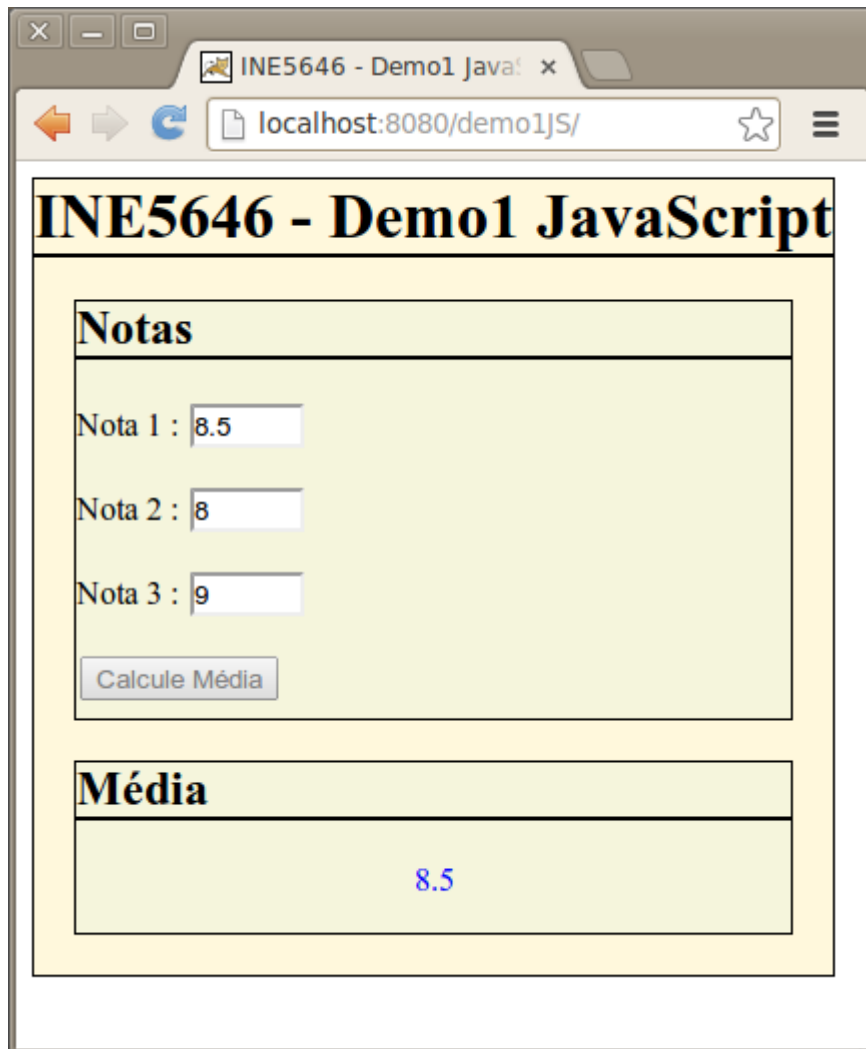
Nota 3 :

**Média**

não definida

# Aplicação Exemplo – index.jsp

- Média que aprova e que reprova.



INE5646 - Demo1 JavaScript

**Notas**

Nota 1 :

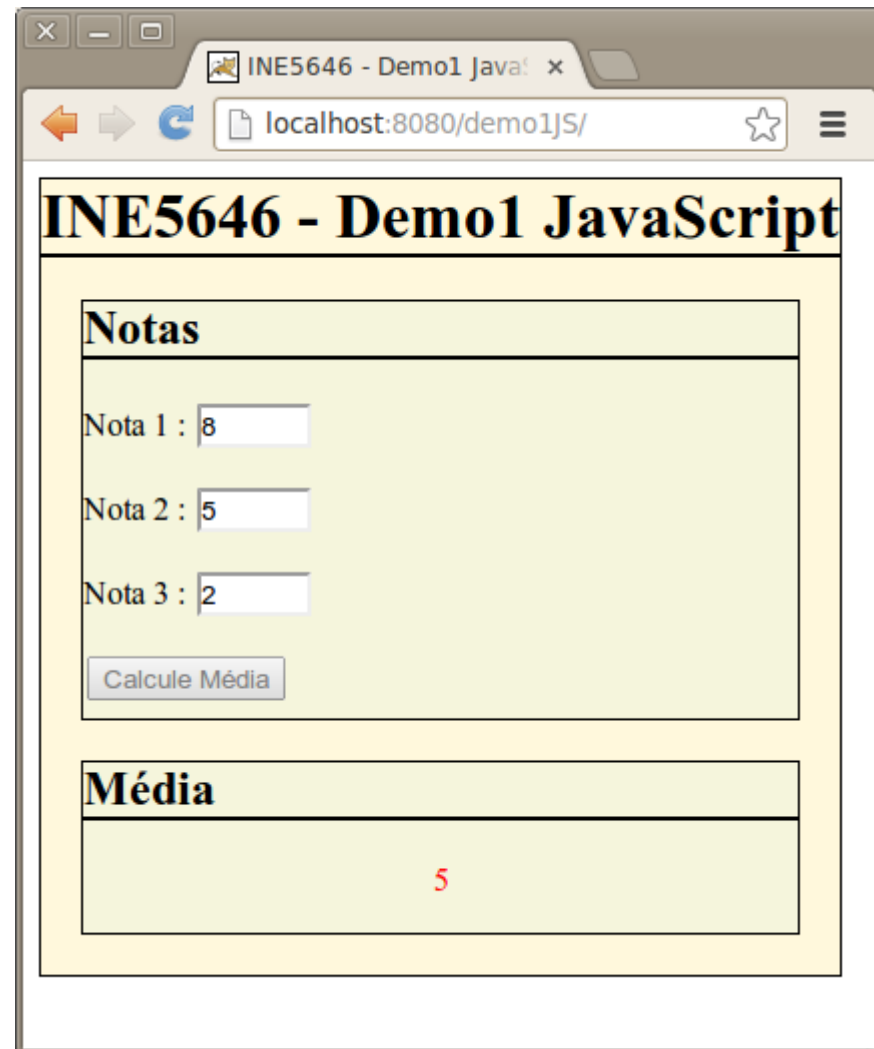
Nota 2 :

Nota 3 :

**Média**

8.5

Detailed description: This screenshot shows the web application interface after a successful calculation. The browser window title is 'INE5646 - Demo1 JavaScript' and the address bar shows 'localhost:8080/demo1JS/'. The page has a yellow background. Under the heading 'INE5646 - Demo1 JavaScript', there is a section titled 'Notas' (Grades) containing three input fields for 'Nota 1', 'Nota 2', and 'Nota 3' with values 8.5, 8, and 9 respectively. Below these is a 'Calcule Média' button. At the bottom, a section titled 'Média' (Average) displays the result '8.5' in blue text.



INE5646 - Demo1 JavaScript

**Notas**

Nota 1 :

Nota 2 :

Nota 3 :

**Média**

5

Detailed description: This screenshot shows the web application interface after a failed calculation. The browser window title is 'INE5646 - Demo1 JavaScript' and the address bar shows 'localhost:8080/demo1JS/'. The page has a yellow background. Under the heading 'INE5646 - Demo1 JavaScript', there is a section titled 'Notas' (Grades) containing three input fields for 'Nota 1', 'Nota 2', and 'Nota 3' with values 8, 5, and 2 respectively. Below these is a 'Calcule Média' button. At the bottom, a section titled 'Média' (Average) displays the result '5' in red text.

# Aplicação Exemplo – index.jsp

- O usuário deve digitar números e estes devem estar dentro do intervalo [0, 10].

INE5646 - Demo1 JavaScript

**Notas**

Nota 1 : oito

Nota 2 : 5

Nota 3 : 2

Calcule Média

Digite apenas números

**Média**

não definida

INE5646 - Demo1 JavaScript

**Notas**

Nota 1 : 12

Nota 2 : 5

Nota 3 : 2

Calcule Média

As notas devem estar entre 0 e 10

**Média**

não definida

# Aplicação Exemplo – index.jsp

- A página utiliza estilos CSS [linha 6]

```
index.jsp x
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <link rel="stylesheet" href="css/estilos.css">
7     <title>INE5646 - Demo1 JavaScript</title>
8   </head>
```



# Aplicação Exemplo – index.jsp

- Atente para as tags HTML com atributo **id** definindo a estrutura da interface com o usuário.

```
9      <body>
10      <div id="app">
11          <h1>INE5646 - Demo1 JavaScript</h1>
12          <div id="entrada" class="painel">
13              <div id="notas">
14                  <h2>Notas</h2>
15                  <p>Nota 1 : <input id="n1" type="text" size="4"></p>
16                  <p>Nota 2 : <input id="n2" type="text" size="4"></p>
17                  <p>Nota 3 : <input id="n3" type="text" size="4"></p>
18              </div>
19              <input type="submit" id="btCalcular" value="Calcule Média">
20              <div id="msg"></div>
21          </div>
22
23          <div id="saida" class="painel">
24              <h2>Média</h2>
25              <p id="media">não definida</p>
26          </div>
27      </div>
28  
```

# Aplicação Exemplo – index.jsp

- No final do arquivo, a tag script faz com que o código JavaScript armazenado no servidor no arquivo demo1.js seja baixado e executado.

```
28  
29     <script src="js/demo1.js"></script>  
30 </body>  
31 </html>
```

# Aplicação Exemplo – estilos.css

- Estilos CSS usados na aplicação.

```
estilos.css
1  #app {
2      border: solid 1px;
3      display: inline-block;
4      background-color: cornsilk;
5  }
6
7  #entrada,#saida {
8      border: solid 1px;
9  }
10
11 #msg {
12     background-color: black;
13     color: lime;
14     margin: 2%;
15 }
16
17 #media {
18     text-align: center;
19 }
20
```

```
20
21 h1,h2 {
22     margin-top: 0px;
23     border-bottom: solid 2px;
24 }
25
26 .painel {
27     background-color: beige;
28     margin: 5%;
29 }
30
31 .aprovado {
32     color: blue;
33 }
34
35 .reprovado {
36     color: red;
37 }
```

# Aplicação Exemplo – demo1.js

- A variável **demo1** representa um objeto cujo objetivo é armazenar objetos DOM que serão usados em outras funções (mostradas mais adiante).

```
demo1.js
1  /*
2   * demo1.js
3   */
4
5  // objeto que contém elementos DOM importantes
6  var demo1 = {
7      notas: document.getElementById("notas"),
8      btCalcular: document.getElementById("btCalcular"),
9      n1: document.getElementById("n1"),
10     n2: document.getElementById("n2"),
11     n3: document.getElementById("n3"),
12     pMedia: document.getElementById("media"),
13     msg: document.getElementById("msg")
14 };
15
```

# Aplicação Exemplo – demo1.js

- A função **evNotaAlterada**:
  - É executada sempre que o valor de alguma das notas é alterado. **Programação por eventos!**
  - É o *listener* de algum evento (definido mais adiante).
  - Tem por objetivo verificar se as notas que o usuário digitou são válidas, habilitando assim, o botão que calculará a média das notas.

```
16 // verifica se as notas são válidas
17 function evNotaAlterada() {
18     // obter os 3 valores digitados
19     var vn1 = demo1.n1.value;
20     var vn2 = demo1.n2.value;
21     var vn3 = demo1.n3.value;
22
23     // apaga a mensagem de erro
24     demo1.msg.innerHTML = "";
25     // define o texto atual para média
26     demo1.pMedia.innerHTML = "não definida";
27     // remove qualquer class CSS
28     demo1.pMedia.className = "";
```



# Aplicação Exemplo – demo1.js

- A função evNotaAlterada (cont):
  - Os 3 valores informados são obtidos (linhas 19 a 21).
  - Sempre que uma nota for alterada é preciso apagar as modificações na página que ocorreram em função da alteração de nota anterior (linhas 24, 26 e 28).

```
16 // verifica se as notas são válidas
17 function evNotaAlterada() {
18     // obter os 3 valores digitados
19     var vn1 = demo1.n1.value;
20     var vn2 = demo1.n2.value;
21     var vn3 = demo1.n3.value;
22
23     // apaga a mensagem de erro
24     demo1.msg.innerHTML = "";
25     // define o texto atual para média
26     demo1.pMedia.innerHTML = "não definida";
27     // remove qualquer class CSS
28     demo1.pMedia.className = "";
```

# Aplicação Exemplo – demo1.js

- A função evNotaAlterada (cont):
  - Se pelo menos 1 dos 3 valores informados não for número (linha 31) então o botão para calcular a média deve ser desabilitado (linha 33) e uma mensagem deve ser mostrada ao usuário (linha 35).

```
30 // se algum dos campos não for número então
31 if (isNaN(vn1) || isNaN(vn2) || isNaN(vn3)) {
32     // desabilitar o botão que calcula a média
33     demo1.btCalcular.disabled = true;
34     // avisa o usuário que deve digitar apenas números
35     demo1.msg.innerHTML = "Digite apenas números";
36 } else {
37     // obtém os 3 números digitados
38     var n1 = new Number(vn1);
39     var n2 = new Number(vn2);
40     var n3 = new Number(vn3);
41     if (n1 < 0 || n1 > 10 ||
42         n2 < 0 || n2 > 10 ||
43         n3 < 0 || n3 > 10) {
44         // se algum deles estiver fora do intervalo [0, 10]
45         demo1.btCalcular.disabled = true;
46         demo1.msg.innerHTML = "As notas devem estar entre 0 e 10";
47     }
```

# Aplicação Exemplo – demo1.js

- A função evNotaAlterada (cont):
  - Se pelo menos 1 dos 3 valores informados não estiver no intervalo de 0 a 10 (linhas 38 a 43) então o botão para calcular a média deve ser desabilitado (linha 45) e uma mensagem deve ser mostrada ao usuário (linha 46).

```
30 // se algum dos campos não for número então
31 if (isNaN(vn1) || isNaN(vn2) || isNaN(vn3)) {
32     // desabilitar o botão que calcula a média
33     demo1.btCalcular.disabled = true;
34     // avisa o usuário que deve digitar apenas números
35     demo1.msg.innerHTML = "Digite apenas números";
36 } else {
37     // obtém os 3 números digitados
38     var n1 = new Number(vn1);
39     var n2 = new Number(vn2);
40     var n3 = new Number(vn3);
41     if (n1 < 0 || n1 > 10 ||
42         n2 < 0 || n2 > 10 ||
43         n3 < 0 || n3 > 10) {
44         // se algum deles estiver fora do intervalo [0, 10]
45         demo1.btCalcular.disabled = true;
46         demo1.msg.innerHTML = "As notas devem estar entre 0 e 10";
47     }
```



# Aplicação Exemplo – demo1.js

- A função evNotaAlterada (cont):
  - Caso as condições anteriores sejam falsas então o usuário digitou 3 números válidos e, por isso, o botão que calcula a média deve passar a ser clicável.

```
47     }  
48     else  
49         // habilita o botão que calcula a média  
50         demo1.btCalcular.disabled = false;  
51     }  
52 }  
53
```

# Aplicação Exemplo – demo1.js

- A função **evMediaSolicitada**:
  - Seu objetivo é mostrar a média das notas (linha 69).

```
54 // o usuário solicitou que a média fosse calculada
55 function evMediaSolicitada() {
56     // obter os 3 números
57     var num1 = new Number(demo1.n1.value);
58     var num2 = new Number(demo1.n2.value);
59     var num3 = new Number(demo1.n3.value);
60     var media = (num1 + num2 + num3) / 3;
61     if (media >= 5.75)
62         // adicionar a class 'aprovado' para o valor da média
63         demo1.pMedia.className = "aprovado";
64     else
65         // adicionar a class 'reprovado' para o valor da média
66         demo1.pMedia.className = "reprovado";
67
68     // definir o valor da média
69     demo1.pMedia.innerHTML = media;
70
71     // não precisa mais calcular o valor da média
72     demo1.btCalcular.disabled = true;
73 }
```

# Aplicação Exemplo – demo1.js

- A função init:
  - Seu objetivo é associar funções a eventos (linhas 81 e 84).
  - Linha 91 : executa a função init.

```
75 // inicialização, associando funções à eventos
76 function init() {
77     // se a div notas receber o evento "change" então executará a função
78     // evNotaAlterada. Observar que a função está associada à div notas e
79     // não a cada um dos inputs. O evento "change" é gerado em algum input
80     // e sobe (bubbling) na hierarquia DOM.
81     demo1.notas.addEventListener("change", evNotaAlterada);
82
83     // outra forma de associar funções à eventos
84     demo1.btCalcular.onclick = evMediaSolicitada;
85
86     // inicialmente o botão de cálculo de média fica desabilitado
87     demo1.btCalcular.disabled = true;
88 }
89
90 // invoca a função init
91 init();
```

# Aplicação Exemplo – demo1.js

- O arquivo demo1.js utiliza objetos DOM diretamente. Exemplos (em vermelho):
  - demo1.notas.**addEventListener**("change", f);
  - demo1.btCalcular.**onclick**(f);
  - demo1.pMedia.**className** = "aprovado";
  - demo1.pMedia.**innerHTML** = media;
- O problema é que a especificação DOM não é seguida/implementada da mesma maneira em cada navegador. **Não há garantia, portanto, que as funções JavaScript serão executadas corretamente.**

# Bibliotecas Cross-Browser

- Para o desenvolvedor de aplicações, tratar a incompatibilidade dos diversos navegadores criando uma versão das funções JavaScript para cada (versão de) navegador é uma tarefa tediosa e sempre sujeita a erros.
- A solução:
  - Deixar que outros façam isso!
  - Criar uma biblioteca de funções JavaScript que, na sua implementação, trate as diferenças entre os navegadores. Estas bibliotecas são chamadas de **cross-browser**.
- **jQuery** (<http://jquery.com>) é o exemplo mais famoso de biblioteca cross-browser.

# jQuery

- Além da questão cross-browser, torna a programação JavaScript mais simples.





# jQuery

- Alguns exemplos rápidos (fonte: página inicial do jQuery):

## DOM Traversal and Manipulation

Get the `<button>` element with the class 'continue' and change its HTML to 'Next Step...'

```
1 | $( "button.continue" ).html( "Next Step..." )
```

# jQuery

- Alguns exemplos rápidos (fonte: página inicial do jQuery):

## Event Handling

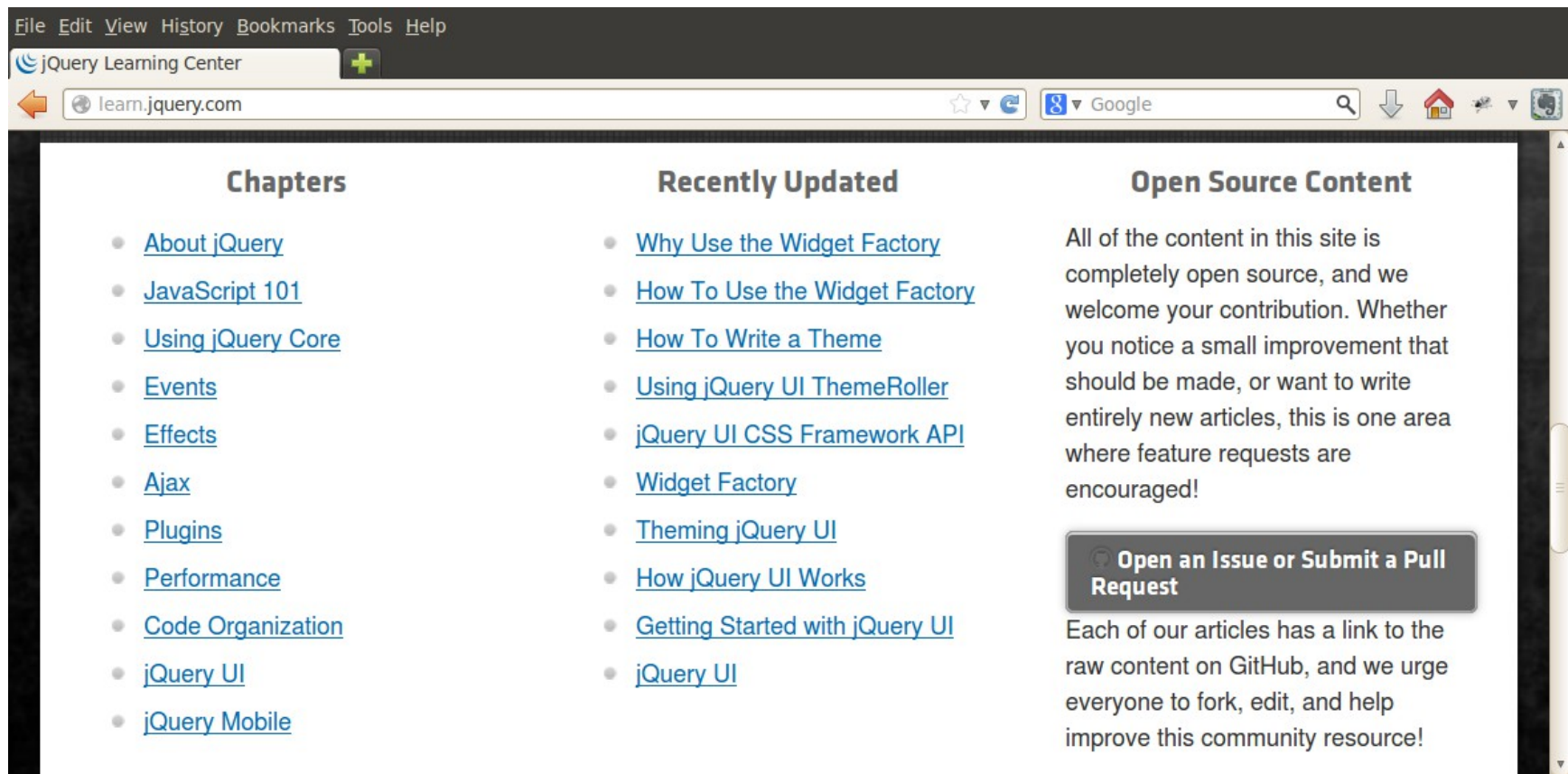
Show the `#banner-message` element that is hidden with `display:none` in its CSS when any button in `#button-container` is clicked.

```
1 | var hiddenBox = $( "#banner-message" );  
2 | $( "#button-container button" ).on( "click", function( event ) {  
3 |     hiddenBox.show();  
4 | });
```



# jQuery

- Aprender jQuery: <http://learn.jquery.com>



# jQuery - Exemplo

- O exemplo a seguir é a versão com jQuery do exemplo demo1JS (calcular a média de 3 notas).
- As únicas alterações em relação à versão demo1JS são:
  - No arquivo index.jsp, para incluir o jQuery.
  - No arquivo demo2.js, onde cada comando que usava DOM diretamente é substituído pelo equivalente em jQuery.

# jQuery – Exemplo – index.jsp

- Linha 7: inclusão da biblioteca jQuery.
  - O arquivo tem aproximadamente 92 KB.

```
index.jsp x
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <link rel="stylesheet" href="css/estilos.css">
7     <script type="text/javascript" src="js/jquery-1.9.1.min.js"></script>
8     <title>INE5646 - Demo2 JavaScript</title>
9   </head>
```

```
29
30     <script src="js/demo2.js"></script>
31   </body>
32 </html>
```

# jQuery – Exemplo – demo2.js

- Expressões que iniciam com “\$” são funções jQuery.
- \$(" #notas") equivale, do ponto de vista lógico, à document.getElementById(“notas”).

```
demo2.js
1  /*
2   * demo2.js
3   */
4
5  // objeto que contém elementos DOM importantes
6  var demo2 = {
7      notas : $("#notas"),
8      btCalcular: $("#btCalcular"),
9      n1: $("#n1"),
10     n2: $("#n2"),
11     n3: $("#n3"),
12     pMedia: $("#media"),
13     msg: $("#msg")
14 };
15
```

# jQuery – Exemplo – demo2.js

- Função evNotaAlterada:

```
16 // verifica se as notas são válidas
17 function evNotaAlterada() {
18     // obter os 3 valores digitados
19     var vn1 = demo2.n1.val();
20     var vn2 = demo2.n2.val();
21     var vn3 = demo2.n3.val();
22
23     // apaga a mensagem de erro
24     demo2.msg.html("");
25
26     // define o texto atual para média
27     demo2.pMedia.html("não definida");
28
29     // remove qualquer class CSS
30     demo2.pMedia.removeClass();
```

# jQuery – Exemplo – demo2.js

- Função evNotaAlterada (cont):

```
31
32 // se algum dos campos não for número então
33 if (isNaN(vn1) || isNaN(vn2) || isNaN(vn3)) {
34     // desabilitar o botão que calcula a média
35     demo2.btCalcular.attr("disabled", false);
36     // avisa o usuário que deve digitar apenas números
37     demo2.msg.html("Digite apenas números");
38 } else {
39     // obtém os 3 números digitados
40     var n1 = new Number(vn1);
41     var n2 = new Number(vn2);
42     var n3 = new Number(vn3);
43     if (n1 < 0 || n1 > 10 ||
44         n2 < 0 || n2 > 10 ||
45         n3 < 0 || n3 > 10) {
46         // se algum deles estiver fora do intervalo [0, 10]
47         demo2.btCalcular.attr("disabled", true);
48         demo2.msg.html("As notas devem estar entre 0 e 10");
49     }
50     else
51         // habilita o botão que calcula a média
52         demo2.btCalcular.attr("disabled", false);
53 }
54 }
```



# jQuery – Exemplo – demo2.js

- Função evMediaSolicitada:

```
56 // o usuário solicitou que a média fosse calculada
57 function evMediaSolicitada() {
58     // obter os 3 números
59     var num1 = new Number(demo2.n1.val());
60     var num2 = new Number(demo2.n2.val());
61     var num3 = new Number(demo2.n3.val());
62     var media = (num1 + num2 + num3) / 3;
63     if (media >= 5.75)
64         // adicionar a class 'aprovado' para o valor da média
65         demo2.pMedia.addClass("aprovado");
66     else
67         // adicionar a class 'reprovado' para o valor da média
68         demo2.pMedia.addClass("reprovado");
69
70     // definir o valor da média
71     demo2.pMedia.html(media);
72
73     // não precisa mais calcular o valor da média
74     demo2.btCalcular.attr("disabled", true);
75 }
```

# jQuery – Exemplo – demo2.js

- Função init:

```
77 // inicialização, associando funções à eventos
78 function init() {
79     // se a div notas receber o evento "change" então executará a função
80     // evNotaAlterada. Observar que a função está associada à div notas e
81     // não a cada um dos inputs. O evento "change" é gerado em algum input
82     // e sobe (bubbling) na hierarquia DOM.
83     demo2.notas.on("change", evNotaAlterada);
84
85     demo2.btCalcular.on("click", evMediaSolicitada);
86
87     // inicialmente o botão de cálculo de média fica desabilitado
88     demo2.btCalcular.attr("disabled", true);
89 }
90
91 // invoca a função init
92 init();
```



# jQuery – Exemplo com Ajax

- O exemplo a seguir mostra o uso da tecnologia Ajax disponível no jQuery.
- O objetivo da aplicação exemplo é mostrar os dados (nome e e-mail) associados a um login de usuário.
- O usuário digita um login e, via requisição Ajax, os dados são obtidos no servidor (camada 2). Quando a resposta chega os dados são exibidos em uma tabela.
- Como Ajax usa **requisição assíncrona**, a interface com o usuário não fica congelada aguardando a resposta.

# jQuery – Exemplo com Ajax

- A medida que o usuário vai digitando o login este vai aparecendo em “Resultado”.

INE5646 - Exemplo Ajax

Dica: experimente fulano e siclano

**Pesquisar Usuários**

Login

Login

**Resultado**

Pesquisar :

Pesquisar :

INE5646 - Exemplo Ajax

Dica: experimente fulano e siclano

**Pesquisar Usuários**

Login fulan

Login

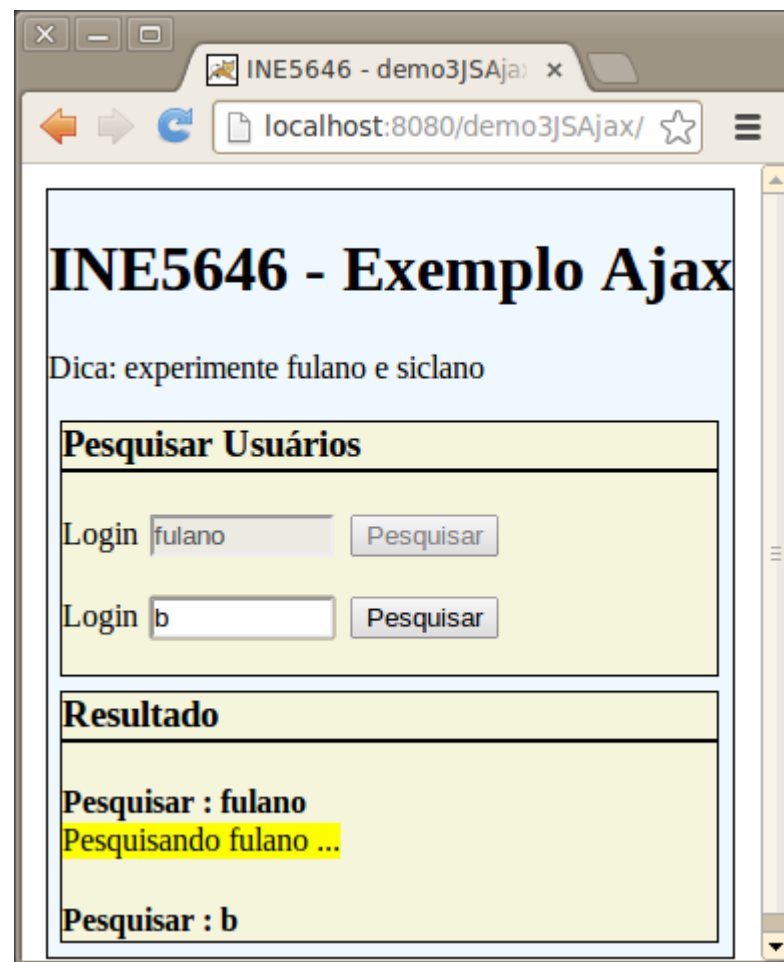
**Resultado**

Pesquisar : fulan

Pesquisar :

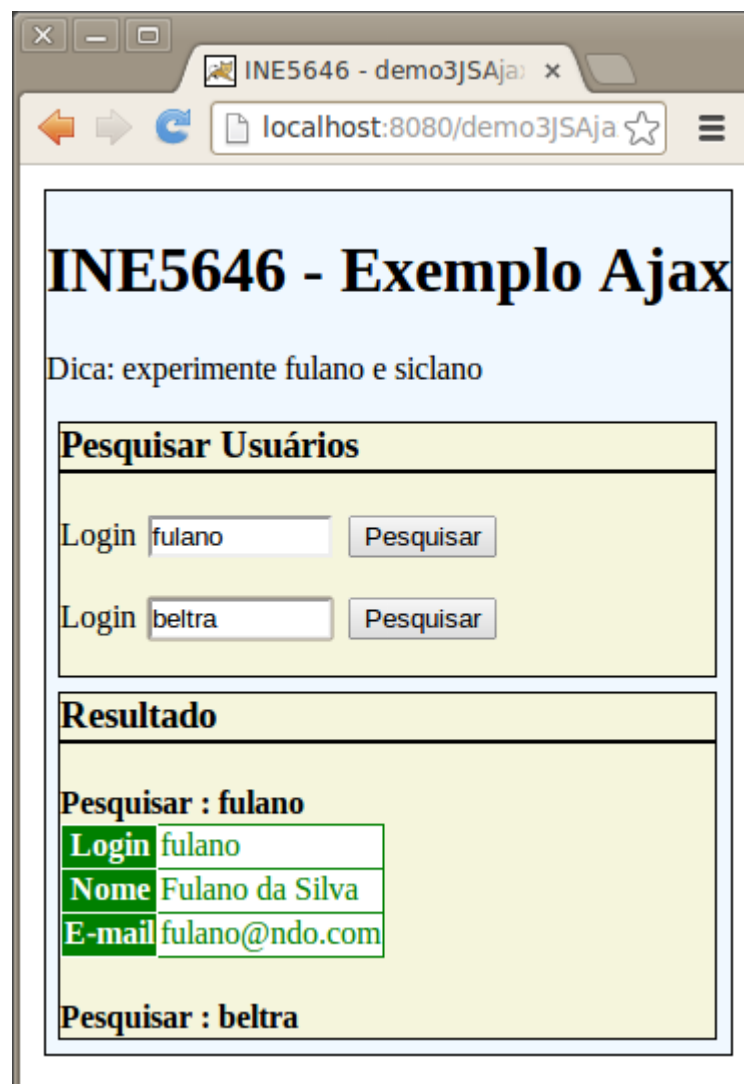
# jQuery – Exemplo com Ajax

- Após ter clicado no botão “Pesquisar”, a requisição ajax é enviada. Enquanto a resposta não chega, o usuário está livre para ir digitando outra pesquisa (na figura, login iniciando com “b”).



# jQuery – Exemplo com Ajax

- Enquanto está digitando a pesquisa “beltra”, a resposta da pesquisa “fulano” chega e é exibida na forma de tabela.



INE5646 - Exemplo Ajax

Dica: experimente fulano e siclano

**Pesquisar Usuários**

Login

Login

**Resultado**

Pesquisar : fulano

Login	fulano
Nome	Fulano da Silva
E-mail	fulano@ndo.com

Pesquisar : beltra

# jQuery – Exemplo com Ajax

- A pesquisa pelo login “beltrano” mostra, após processamento no servidor, que não há nenhum usuário com este login.

INE5646 - Exemplo Ajax

Dica: experimente fulano e siclano

**Pesquisar Usuários**

Login

Login

**Resultado**

**Pesquisar : fulano**

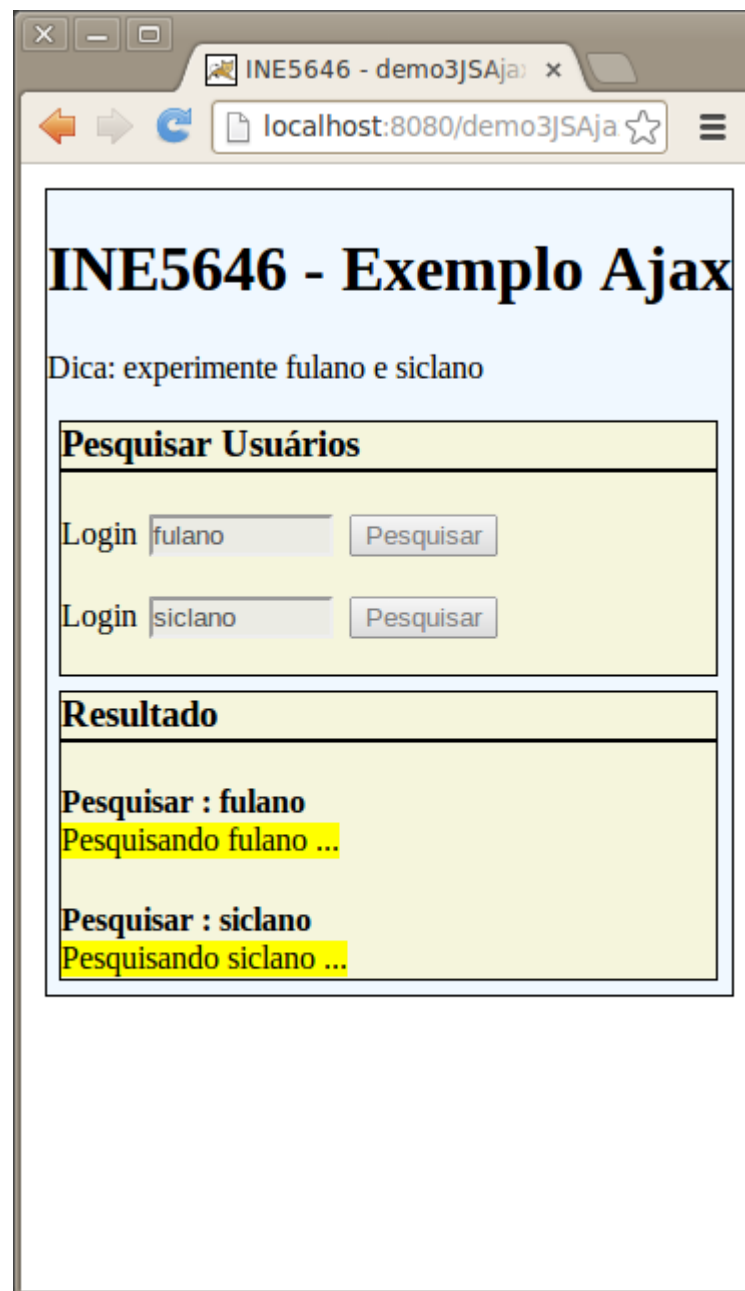
Login	fulano
Nome	Fulano da Silva
E-mail	fulano@ndo.com

**Pesquisar : beltrano**

Nenhum usuário encontrado para login: beltrano

# jQuery – Exemplo com Ajax

- As pesquisas pelos logins “fulano” e “siclano” são realizadas ao mesmo tempo.
- Cada pesquisa é realizada por meio de uma requisição Ajax própria.
- **Mas atenção:** a especificação HTTP (seção 8.1.4) recomenda no máximo 2 requisições simultâneas. Se houvesse mais, elas seriam colocadas em uma fila.



# jQuery – Exemplo com Ajax

- As pesquisas pelos logins “fulano” e “siclano” mostram, após processamento no servidor, seus resultados.

INE5646 - Exemplo Ajax

Dica: experimente fulano e siclano

**Pesquisar Usuários**

Login

Login

**Resultado**

**Pesquisar : fulano**

Login	fulano
Nome	Fulano da Silva
E-mail	fulano@ndo.com

**Pesquisar : siclano**

Login	siclano
Nome	Siclano da Silva
E-mail	siclano@abarte.com

# Exemplo com Ajax – index.jsp

- A linha 8 indica o uso da biblioteca jQuery.
- A linha 9 indica o uso de código JavaScript específico da aplicação.

```
index.jsp x
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6     <title>INE5646 - demo3JSAjax - Ajax</title>
7     <link rel="stylesheet" href="css/estilos.css">
8     <script type="text/javascript" src="js/jquery-1.9.1.min.js"></script>
9     <script type="text/javascript" src="js/demo3JSAjax.js"></script>
10  </head>
```



# Exemplo com Ajax – index.jsp

- Observe que o código HTML define apenas a estrutura (diversas divs) e o conteúdo da página.

```
11  <body>
12    <div id="app">
13      <h1>INE5646 - Exemplo Ajax</h1>
14      <p>Dica: experimente fulano e siclano</p>
15      <div id="entrada" class="painel">
16        <h3>Pesquisar Usuários</h3>
17        <p>Login <input type="text" id="c1" size="10" name="c1">
18          <input type="submit" id="btC1" value="Pesquisar"></p>
19        <p>Login <input type="text" id="c2" size="10" name="c2">
20          <input type="submit" id="btC2" value="Pesquisar"></p>
21      </div>
22      <div id="saida" class="painel">
23        <h3>Resultado</h3>
24        <h4 id="login1">Pesquisar :</h4>
25        <div id="r1"></div>
26
27        <h4 id="login2">Pesquisar :</h4>
28        <div id="r2"></div>
29      </div>
30    </div>
```

# Exemplo com Ajax – index.jsp

- O código JavaScript abaixo garante que o processamento definido pela função init só será executado após toda a página ter sido carregada.

```
31
32     <script type="text/javascript">
33         $(document).ready(function() {
34             init();
35         });
36     </script>
37 </body>
38 </html>
```

# Exemplo com Ajax – estilos.css

```
estilos.css
1 #app {
2     border: solid 1px;
3     margin: 1%;
4     display: inline-block;
5     background-color: aliceblue;
6 }
7
8 .painel {
9     border: solid 1px;
10    margin: 2%;
11    background-color: beige;
12 }
13
14 .aviso {
15     background-color: yellow;
16 }
17
18 .ok {
19     color: green;
20 }
21
22 .nok {
23     color: lime;
24     background-color: black;
25 }
26
27 .erro {
28     color: red;
29 }
```

```
30
31 h3 {
32     margin-top: 0px;
33     border-bottom: solid 2px;
34 }
35
36 h4 {
37     margin-bottom: 0px;
38 }
39
40 table {
41     border: solid 1px;
42     border-collapse: collapse;
43 }
44
45 th {
46     border: solid 1px;
47     background-color: green;
48     color: white;
49 }
50
51 td {
52     border: solid 1px;
53     background-color: white;
54     color: green;
55 }
```

# Exemplo com Ajax – Usuario.java

```
Usuario.java x
1 package demo3jsajax.dados;
2
3 public class Usuario {
4     String login;
5     String nome;
6     String email;
7
8     public Usuario(String login, String nome, String email) {
9         this.login = login;
10        this.nome = nome;
11        this.email = email;
12    }
13
14    public String getLogin() {
15        return login;
16    }
17
18    public String getNome() {
19        return nome;
20    }
21
22    public String getEmail() {
23        return email;
24    }
25 }
```

# Exemplo com Ajax – BD.java

```
BD.java
1 package demo3jsajax.dados;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class BD {
7     static Map<String, Usuario> dados;
8
9     static void crieDados() {
10         dados = new HashMap<String, Usuario>();
11         dados.put("fulano", new Usuario("fulano", "Fulano da Silva", "fulano@ndo.com"));
12         dados.put("siclano", new Usuario("siclano", "Siclano da Silva", "siclano@abarte.com"));
13     }
14
15     public static Usuario pesquise(String login) {
16         if (dados == null)
17             crieDados();
18         return dados.get(login);
19     }
20
21 }
```



# Exemplo com Ajax – ServletPesquisador.java

```
19 @WebServlet(name = "ServletPesquisador", urlPatterns = {"/pesquiseUsuario"})
20 public class ServletPesquisador extends HttpServlet {
```

```
32     protected void processRequest(HttpServletRequest request,
33                                   HttpServletResponse response)
34         throws ServletException, IOException {
35         response.setContentType("application/json;charset=UTF-8");
36         PrintWriter out = response.getWriter();
37         try {
38             String login = request.getParameter("login");
39             Usuario u = BD.pesquise(login);
40             String resposta;
41             if (u == null) {
42                 resposta = toJSON(new String[]{"cod", "NOK"});
43             } else {
44                 String uJSON = toJSON(new String[]{
45                     "login", u.getLogin(),
46                     "nome", u.getNome(),
47                     "email", u.getEmail()});
48
49                 resposta = toJSON(new String[]{"cod", "OK", "usuario", uJSON});
50             }
51             try {
52                 Thread.sleep(4000); // para simular uma operação longa
53             } catch (InterruptedException ex) {
54             }
55
56             out.print(resposta);
57         } finally {
58             out.close();
59         }
60     }
```

# Exemplo com Ajax – ServletPesquisador.java

- Método provisório para retornar um objeto JSON a partir de um array de strings.
- Em uma aplicação real, o recomendado é usar alguma biblioteca especializada nesta função.

```
62     private String toJSON(String[] atributoValor) {
63         String a = "\"";
64         StringBuilder sb = new StringBuilder("{");
65         for (int i = 0; i < atributoValor.length; i = i + 2) {
66             sb.append(a).append(atributoValor[i]).append(a).append(":");
67             if (atributoValor[i+1].charAt(0) == '{')
68                 sb.append(atributoValor[i+1]);
69             else
70                 sb.append(a).append(atributoValor[i + 1]).append(a);
71             if (i + 2 < atributoValor.length) {
72                 sb.append(", ");
73             }
74         }
75         return sb.append("}").toString();
76     }
```

# Exemplo Ajax – demo3JSAjax.js

- A função init define funções manipuladoras de eventos (*event handlers*). Exemplo:
  - Linha 2 : Quando o nodo DOM com atributo id igual a “btC1” for clicado então a função “pesquise1” será executada.

```
<div id="entrada" class="painel">
  <h3>Pesquisar Usuários</h3>
  <p>Login <input type="text" id="c1" size="10" name="c1">
    <input type="submit" id="btC1" value="Pesquisar"></p>
  <p>Login <input type="text" id="c2" size="10" name="c2">
```

```
demo3JSAjax.js
1 function init() {
2   $("#btC1").on("click", pesquisa1);
3   $("#btC2").on("click", pesquisa2);
4   $("#c1").on("keyup", atualizeNome1);
5   $("#c2").on("keyup", atualizeNome2);
6 }
7
```



# Exemplo Ajax – demo3JSAjax.js

- A função init define funções manipuladoras de eventos (*event handlers*). Exemplo:
  - Linha 4 : Cada vez que o usuário digitar uma letra do login na tag input com id “c1” a função atualizeNome1 deve ser executada.

```
<div id="entrada" class="painel">
  <h3>Pesquisar Usuários</h3>
  <p>Login <input type="text" id="c1" size="10" name="c1">
    <input type="submit" id="btC1" value="Pesquisar"></p>
  <p>Login <input type="text" id="c2" size="10" name="c2">
```

```
demo3JSAjax.js
1 function init() {
2   $("#btC1").on("click", pesquise1);
3   $("#btC2").on("click", pesquise2);
4   $("#c1").on("keyup", atualizeNome1);
5   $("#c2").on("keyup", atualizeNome2);
6 }
7
```

# Exemplo Ajax – demo3JSAjax.js

- O objetivo da função atualizeNome é mostrar o nome do login que está sendo digitado.

```
<div id="saida" class="painel">
  <h3>Resultado</h3>
  <h4 id="login1">Pesquisar :</h4>
  <div id="r1"></div>

  <h4 id="login2">Pesquisar :</h4>
  <div id="r2"></div>
</div>
```

```
8  function atualizeNome1() {
9    atualizeNome($("#login1"), $("#c1").val(), $("#r1"));
10 }
11
12 function atualizeNome2() {
13   atualizeNome($("#login2"), $("#c2").val(), $("#r2"));
14 }
15
16 function atualizeNome($login, valor, $r) {
17   $login.html("Pesquisar : " + valor);
18   $r.html("");
19 }
```

# Exemplo Ajax – demo3JSAjax.js

```
21 function pesquise1() {
22     pesquise($("#c1"), $("#r1"), $("#btC1"));
23 }
24
25 function pesquise2() {
26     pesquise($("#c2"), $("#r2"), $("#btC2"));
27 }
28
29 function pesquise($c, $r, $btC) {
30     var login = $c.val();
31
32     $r.removeClass();
33     if (login.length === 0) {
34         $r.addClass("erro");
35         $r.html("digite um login!");
36     }
37     else {
38         $r.html(avisar(login));
39         $btC.attr("disabled", true);
40         $c.attr("disabled", true);
41         pesquiseAjax(login, $r, $btC, $c);
42     }
43 }
```

- As funções **pesquise** atualizam a interface, por exemplo, desabilitando o botão de pesquisa (linha 39) para que o usuário não submeta nova pesquisa antes do resultado da pesquisa chegar.
- Por convenção, variáveis que iniciam com “\$” representam objetos jQuery.
- Objetos jQuery contêm, internamente, objetos DOM equivalentes.

```
89 function avisar(login) {
90     return "<span class='aviso'>Pesquisando " + login + " ...</span>";
91 }
```

# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquisaAjax(login, $r, $btC, $c) {  
47     $.ajax({  
48         url: "pesquiseUsuario",  
49         data: {login: login},  
50         type: "GET",  
51         dataType: "json",  
52         success: function(resposta) {  
53             mostreResposta($r, login, resposta);  
54         },  
55         error: function() {  
56             $r.addClass("erro");  
57             $r.html("deu erro");  
58         },  
59         complete: function() {  
60             $btC.attr("disabled", false);  
61             $c.attr("disabled", false);  
62         }  
63     });  
64 }
```

- O método ajax (linha 47) executa uma requisição assíncrona considerando os dados do objeto JSON passado como parâmetro.

# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquiseAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **url:**
  - indica a URL que está sendo enviada ao servidor.
  - No caso, a URL será `http://localhost:8080/demo3JSAjax/pesquiseUsuario`.



# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquisaAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **data:**
  - Indica quais dados serão enviados ao servidor.
  - No caso, o login a ser pesquisado.
- **type:**
  - Indica o método HTTP da requisição.
  - No caso, GET.

# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquiseAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **dataType:**
  - Indica o MIME-TYPE da resposta enviada pelo servidor.
  - No caso, a resposta será convertida em um objeto JSON.

# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquiseAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **success:**
  - Indica qual função deve ser executada pelo browser quando a resposta, sem erros, chegar.
  - No caso, a variável resposta é o objeto JSON enviado pelo servidor contendo os dados do usuário pesquisado.



# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquiseAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **error:**
  - Indica qual função deve ser executada pelo browser caso a resposta seja um erro.
  - No caso, o erro pode ser um código HTTP (como o 404) ou se o objeto JSON estiver mal formado.

# Exemplo Ajax – demo3JSAjax.js

```
46 function pesquiseAjax(login, $r, $btC, $c) {
47     $.ajax({
48         url: "pesquiseUsuario",
49         data: {login: login},
50         type: "GET",
51         dataType: "json",
52         success: function(resposta) {
53             mostreResposta($r, login, resposta);
54         },
55         error: function() {
56             $r.addClass("erro");
57             $r.html("deu erro");
58         },
59         complete: function() {
60             $btC.attr("disabled", false);
61             $c.attr("disabled", false);
62         }
63     });
64 }
```

- **complete:**
  - Indica qual função deve ser executada pelo browser quando a resposta chegar, independentemente de estar correta ou não.
  - No caso, habilita os campos para nova pesquisa.

# Exemplo Ajax – demo3JSAjax.js

- Função mostreResposta :
  - Mostra a resposta enviada pelo servidor.
  - A resposta pode ser {cod: "NOK"} caso não haja usuário com o login pesquisado.

```
66 function mostreResposta($r, login, resposta) {  
67     if (resposta.cod === "NOK") {  
68         $r.addClass("nok");  
69         $r.html("Nenhum usuário encontrado para login: " + login);  
70     }  
71     else {  
72         $r.addClass("ok");  
73         $r.html(monteTabela(resposta.usuario));  
74     }  
75 }
```

# Exemplo Ajax – demo3JSAjax.js

- Função monteTabela :
  - A tabela é montada usando-se o objeto JSON enviado pelo servidor.

```
77 function monteTabela(usuario) {  
78     var tabela;  
79     tabela =  
80         "<table id='tabela'>" +  
81         "<tr><th>Login</th> <td>" + usuario.login + "</td></tr>" +  
82         "<tr><th>Nome</th> <td>" + usuario.nome + "</td></tr>" +  
83         "<tr><th>E-mail</th> <td>" + usuario.email + "</td></tr>" +  
84         "</table>";  
85  
86     return tabela;  
87 }
```

# O método \$.ajax

- O objeto JSON passado como parâmetro no método ajax possui uma grande quantidade de atributos:
  - <http://api.jquery.com/jQuery.ajax/#jQuery-ajax-settings>
- A documentação completa sobre a tecnologia Ajax implementada na jQuery está disponível em:
  - <http://api.jquery.com/category/ajax/>