Modelagem de Hipermídia
Aula3

Soluções para o problema sugerido na aula anterior:

1) com o uso do objeto "out":

```
<html>
<body>
    <p align="center">TABUADA
        <table border=1>
        <tr><td bgcolor=silver width=20>x</td>
      <%
          for (int i=1;i<10;i++)
             out.write("<td bgcolor=silver width=20>"+i+"</td>");
           out.write("</tr>");
           for (int i=1;i<10;i++) {
             out.write("<tr><td bgcolor=silver>"+i+"</td>");
             for (int j=1;j<10;j++)
             out.write("<td>"+(i*j)+"</td>");
             out.write("</tr>");
           }
        %>
        </table>
    </p>
</body>
</html>
```

2) sem o uso do objeto "out":

```
<html>
<body>
    <p align="center">TABUADA
        <table border=1>
            <tr><td bgcolor=silver width=20>x</td>
            <% for (int i=1;i<10;i++) { %>
            <td bgcolor=silver width=20><%= i %></td>
            <% } %>
            </tr>
            <% for (int i=1;i<10;i++) { %>
            <tr><td bgcolor=silver><%= i %></td>
            <% for (int j=1;j<10;j++) { %>
            <td> <%= (i*j) %> </td>
            <% } %>
            </tr>
            <% } %>
        </table>
    </p>
</body>
</html>
```

Querendo comentar algum trecho, basta colocá-lo entre <%-- e --%> (se estiver dentro do código Java, pode-se usar o padrão da linguagem: /* e */ ou // após uma linha).

Outro objeto útil é o "`request`". Um "request" se refere a uma transação entre o navegador e o servidor. Ao entrar na página gerada, o navegador envia um "request" ao servidor, que retorna o resultado. Como parte desse "request", pode haver muitos dados, inclusive informação que o usuário preencheu em "inputs" de "forms" HTML, por exemplo (aí é que começamos a nos comunicar com o usuário, ou melhor, o usuário com o programa). Com o "`request`", também podemos obter informações como o nome ou IP da máquina cliente. Vejamos o exemplo abaixo:

```
<html>
<body>
  O endereço IP é: <%=request.getRemoteHost()%>
</body>
</html>
```

Mais um objeto interessante é o "response", que afeta a resposta enviada ao cliente. Veja um exemplo que faz o cliente abrir outra página (não é o servidor que redireciona, mas a máquina cliente, que recebe a resposta e chama a nova página):

```
<html>
<body>
<%
   response.sendRedirect("http://www.sun.com");
%>
</body>
</html>
```

Diretivas

Diretivas são comandos acessados com a tag <%@. Um exemplo é a diretiva “page” para lidar com *imports*. A sintaxe dos *imports* no JSP é um pouco diferente do que estamos acostumados, veja exemplo:

```
<%@ page import="java.util.*" %>
<html>
<body>
  <%
    Date data = new Date();
  %>
  Data e hora atuais: <%= data %>
</body>
</html>
```

A diretiva “page” contém uma lista de “imports” separados por vírgulas dentro de aspas, como em <%@ page import="java.util.*,java.text.*" %>.

Outra diretiva importante é a “include” que é usada para incluir fisicamente outro arquivo (HTML, JSP, etc.) no atual. O resultado é como se o arquivo contivesse tudo do outro digitado dentro. No exemplo abaixo, após a “Página 1” virá o conteúdo completo da página “outra.html” (que, é bom lembrar, não foi criada ainda):

```
<html>
<body>
  Página 1<br>
  <%@ include file="outra.html" %>
</body>
</html>
```

Crie a página outra.html no próprio NetBeans (Arquivo-Novo Arquivo-Web-HTML), chame de “outra”, escreva alguma coisa dentro do “body”, salve e rode o exemplo anterior.

Vamos forçar um erro (o que talvez já tenha acontecido antes, sem querer...) para vermos a resposta do Tomcat. Altere o exemplo acima e troque o nome da página a chamar para xxx.html (que não existe). Force outros erros. Rode e analise o resultado dos erros - no NetBeans, no log do Tomcat (nas guias inferiores da tela) e no *browser*.

Declarações

Como vimos, você pode fazer declarações de variáveis ou objetos locais dentro das *tags* <% e %>. Mas, se quiser usar declarações, também pode fazê-lo com `<%!` e `%>`, como abaixo:

```
<html>
<body>
  <%! String s="olá!"; %>
  <%=s%>
</body>
</html>
```

Há, no entanto, um pequeno detalhe: declarações desse tipo viram membros da classe do *Servlet* e só serão avaliadas uma vez na carga da página, portanto é bom evitar usar variáveis desse jeito (a menos que a idéia seja justamente essa), pois várias requisições de uma mesma página no servidor usam a mesma instância da classe (algo parecido com o que ocorre com DLLs em outras arquiteturas como ISAPI e NSAPI), como se fosse a mesma variável para todas as *threads*. Se for preciso usar, deve-se sincronizar o acesso às variáveis nas várias *threads*[1]. Importante: variáveis declaradas dentro do escopo do <% e %> não sofrem esse problema, pois são locais, como em <% i=1; %>.

[1] Um pouco confuso? Realmente, programação *multi-thread* com sincronismo é um tópico que não está no escopo desta disciplina. Simplificando muito, trata da execução de tarefas paralelas dentro de um mesmo processo. Quem quiser mais alguma informação a esse respeito, pode me procurar.