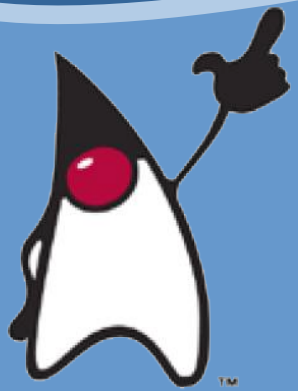
Java Web

MVC – Pattern Commander

Daves Martins

davesmartins@yahoo.com.br

Mestre em Computação de Alto Desempenho pela UFRJ
Especialista em Banco de Dados
Analista Web



Agenda

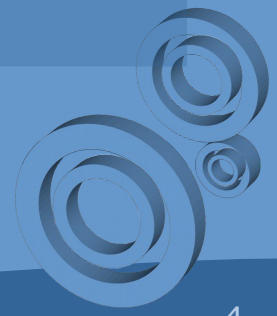
- Visão geral da Arquitetura MVC
- MVC
- Patterns: Command

Boas Práticas

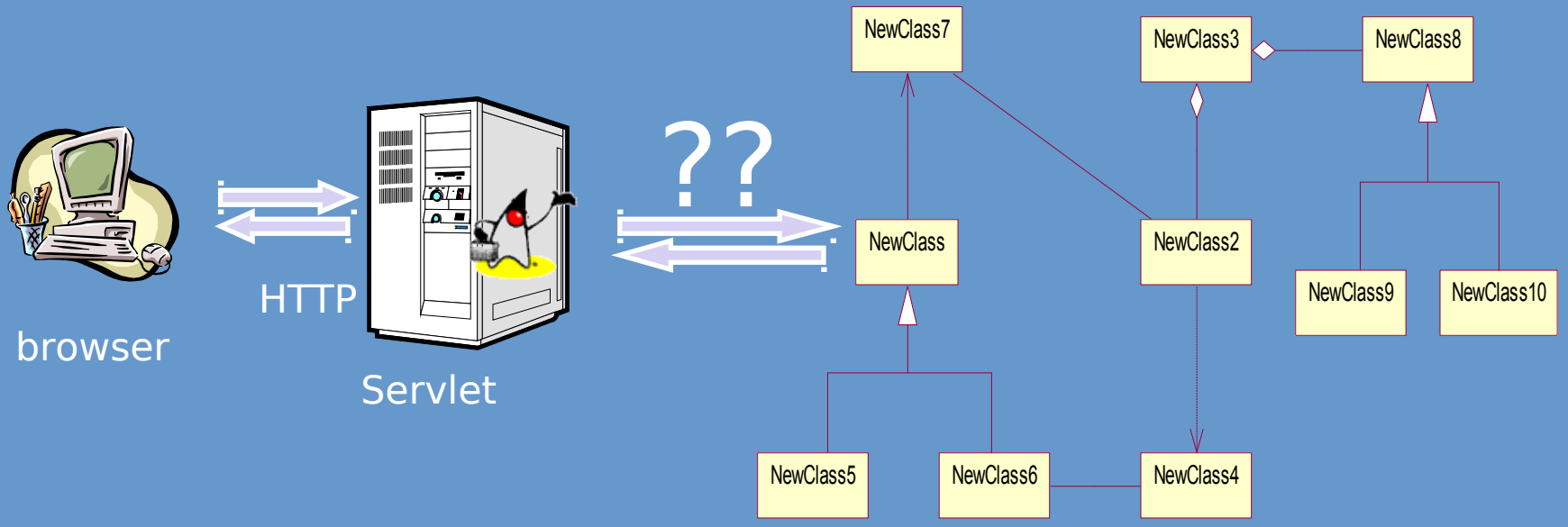
MVC

Servlet + Modelo de Objetos

- Agora sabemos
  - Programar servlets
  - Modelar um sistema orientado a objetos
- Mas como fazer os dois funcionarem juntos?



Servlet + Modelo de Objetos



Arquitetura MVC

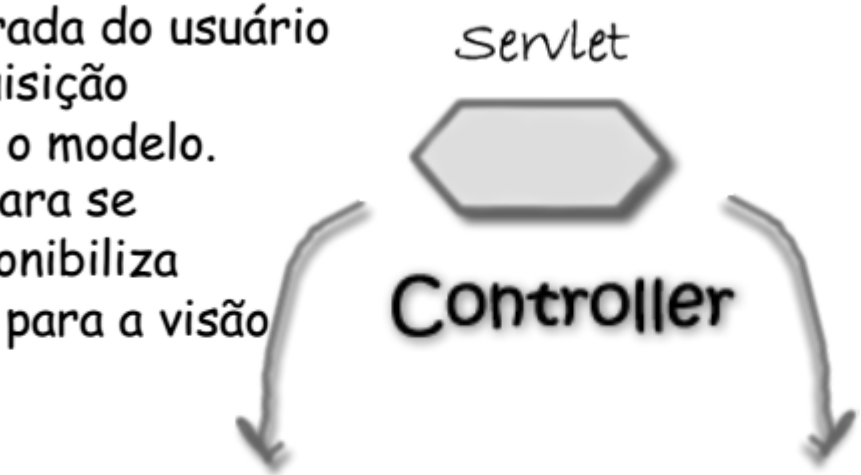
- O que é MVC?
  - MVC – Model-View-Controller
  - Surgiu nos anos 80 com SmallTalk
- Sugere a divisão de uma aplicação visual em 3 partes fundamentais
  - Model
    - Representa o modelo da sua aplicação, com as regras de negócio (business logic) e todo o processamento da aplicação
  - View
    - Representa a informação e recolhe os dados fornecidos pelo usuário
  - Controller
    - Recebe as informações da entrada e as transmite para o modelo

O padrão Model-View-Controller

CONTROLLER (Controlador)

Servlets e JSPs

Recupera a entrada do usuário a partir da requisição e o traduz para o modelo. Diz ao modelo para se atualizar e disponibiliza um novo estado para a visão

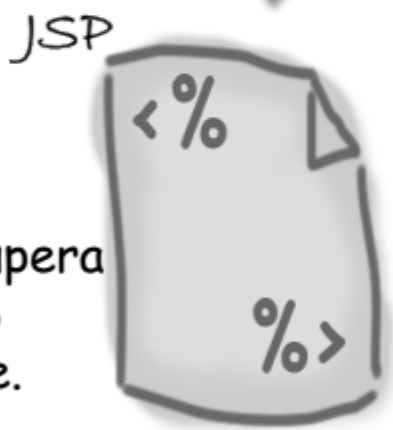


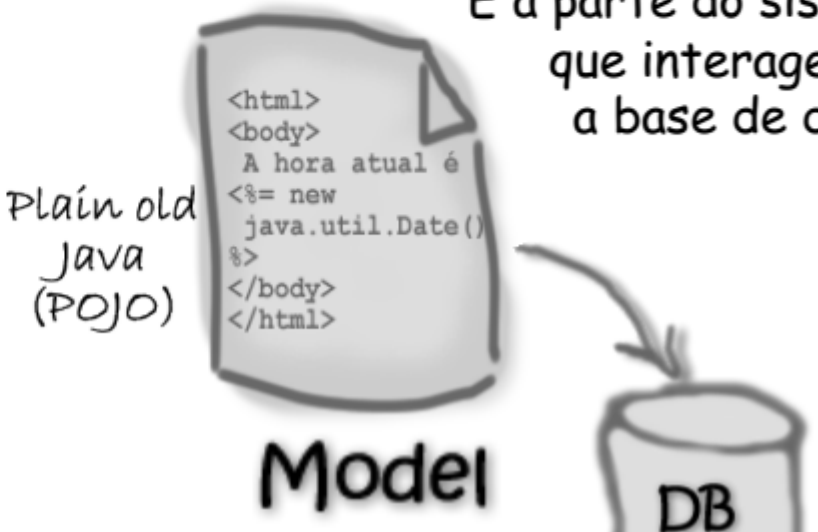
MODEL (Modelo)

Guarda a lógica e o estado do negócio. Conhece as regras para recuperar e alterar o estado. É a parte do sistema que interage com a base de dados

VIEW (Visão)

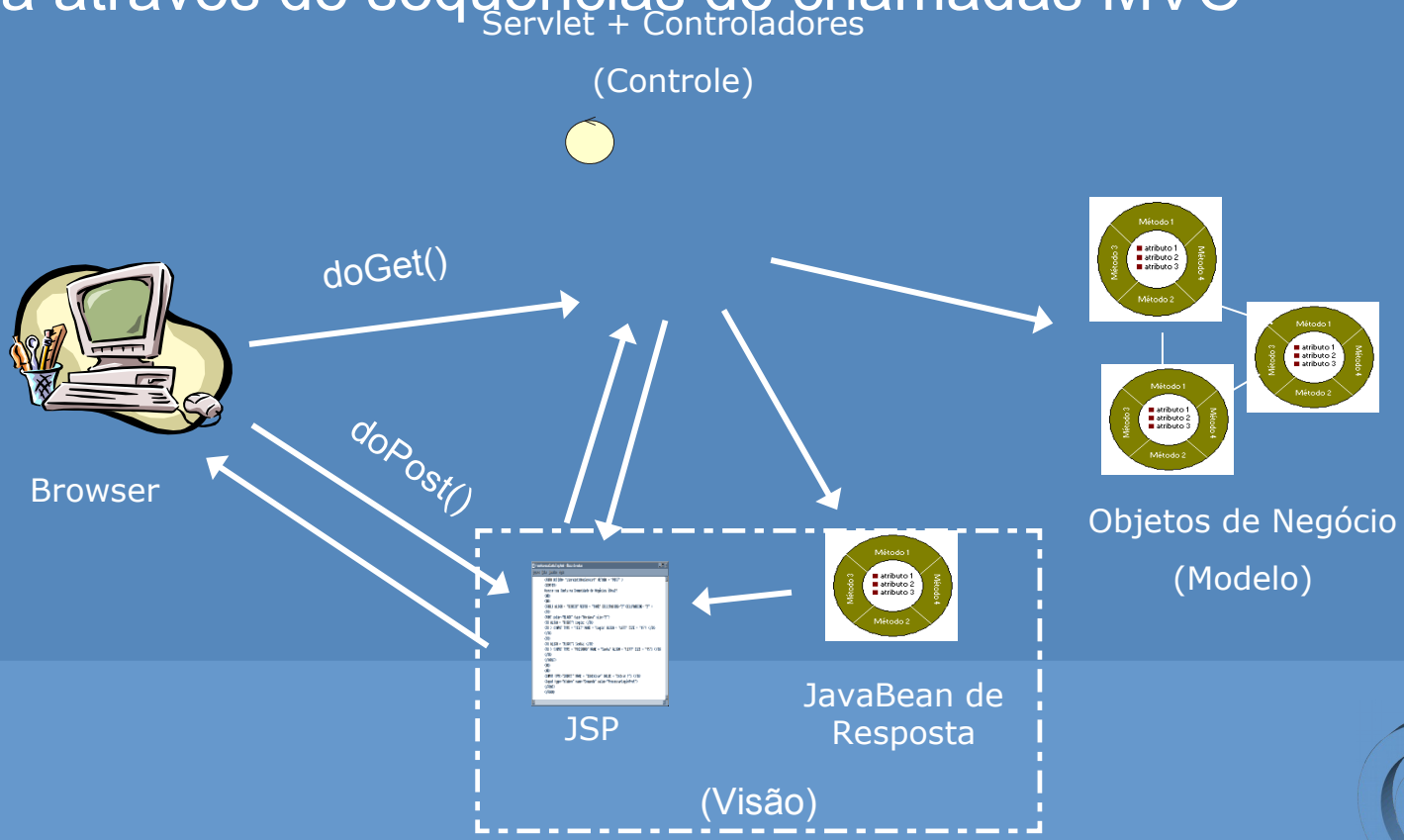
Responsável pela apresentação. Recupera o estado do modelo através do controle. Também obtém a entrada do usuário e repassa para o





Arquitetura MVC

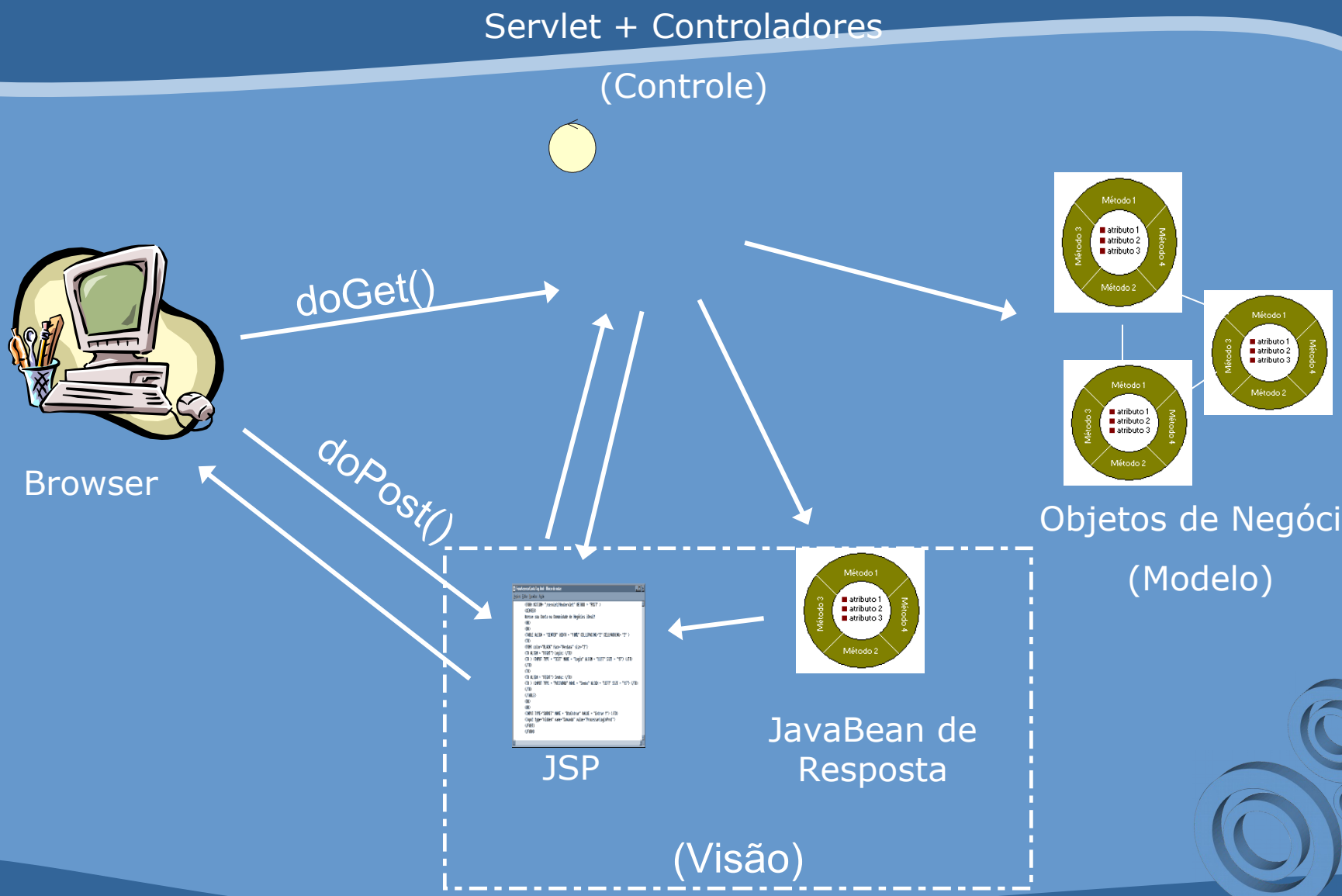
- Pela arquitetura, a interação de um usuário com o sistema se dá através de seqüências de chamadas MVC



Arquitetura MVC

- O modelo será uma aplicação Java orientada a objetos
- O controlador será alguma tecnologia de implementação (padrão comando) que juntamente com o Servlet redirecionará as chamadas feitas pelo cliente aos objetos responsáveis no modelo
- A apresentação será representada pelo JSP

MVC


Servlet + Controladores
(Controle)
doGet()
doPost()
Browser
Método 1
Método 2
Método 3
Método 4
atributo 1
atributo 2
atributo 3
Objetos de Negócio
(Modelo)
JSP
JavaBean de
Resposta
(Visão)

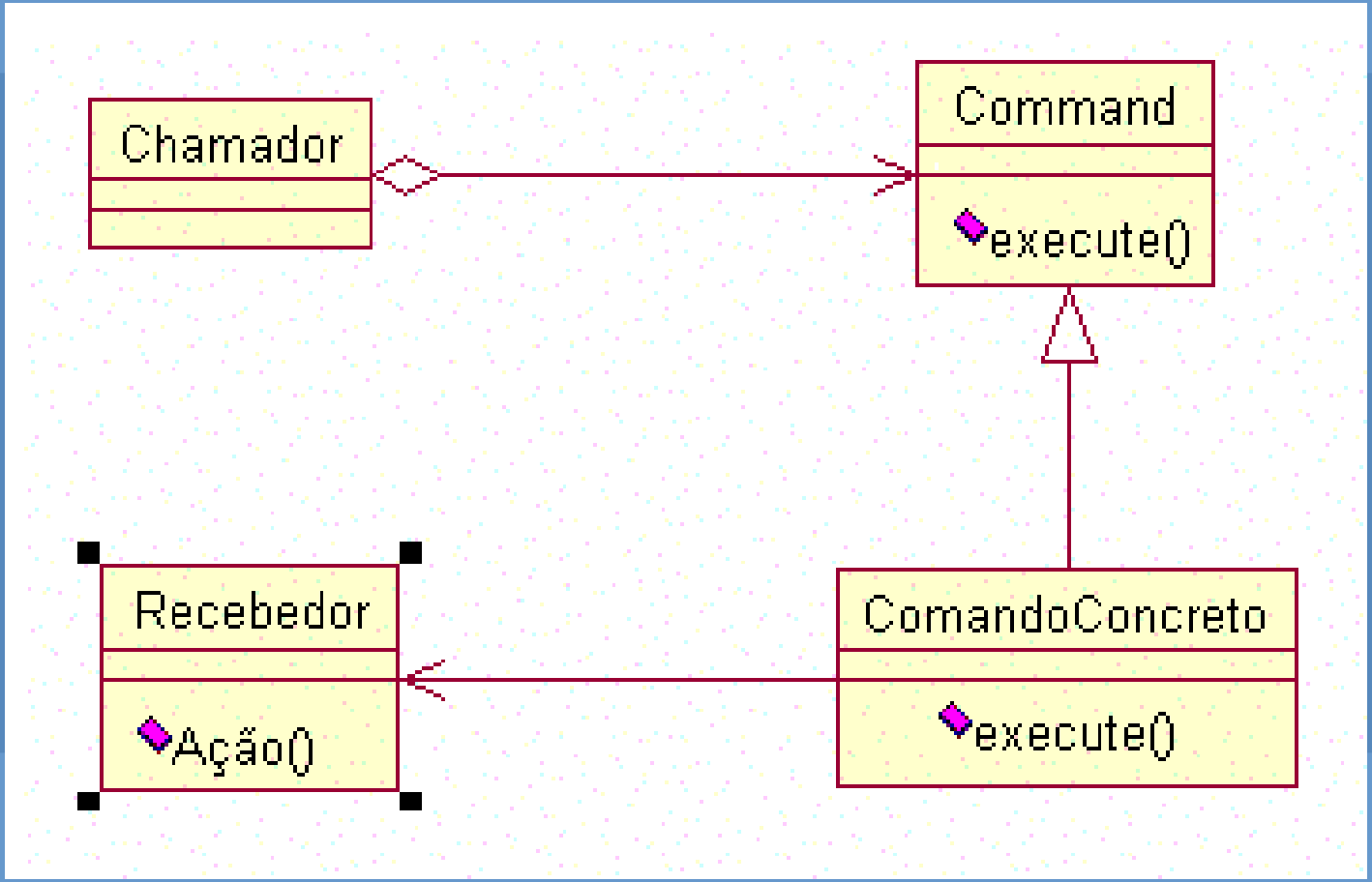
Boas Práticas

Padrão Command

Padrão *Command*

- Objetivos
  - Reduzir acoplamento entre as requisições dos clientes e os objetos que as executam
  - Parametrizar objetos por uma ação a ser executada
  - Especificar, enfileirar e executar solicitações em tempos diferentes para desfazer operações, por exemplo
  - Estruturar um sistema em torno de operações de alto nível, como transações, por exemplo

Padrão *Command*



Padrão *Command* (I)

```
<form method="POST" action="/servlet/ServletCarrinho">
  <p>Carrinho de Brinquedo <br> R$ 10,00</p>
  <input type="hidden" name="Comando" value="Comprar">
  <p><input type="submit" value="Comprar" name="B1"></p>
</form>

<form method="POST" action="/servlet/ServletCarrinho">
  <input type="hidden" name="Comando" value="Fechar">
  <p><input type="submit" value="Fechar Compras" name="B1"></p>
</form>

<form method="POST" action="/servlet/ServletCarrinho">
  <input type="hidden" name="Comando" value="Logout">
  <p><input type="submit" value="Logout" name="B2"></p>
</form>
```

Padrão *Command* (II)

- Como implementar?
  - Atributo hidden
    - Ex:

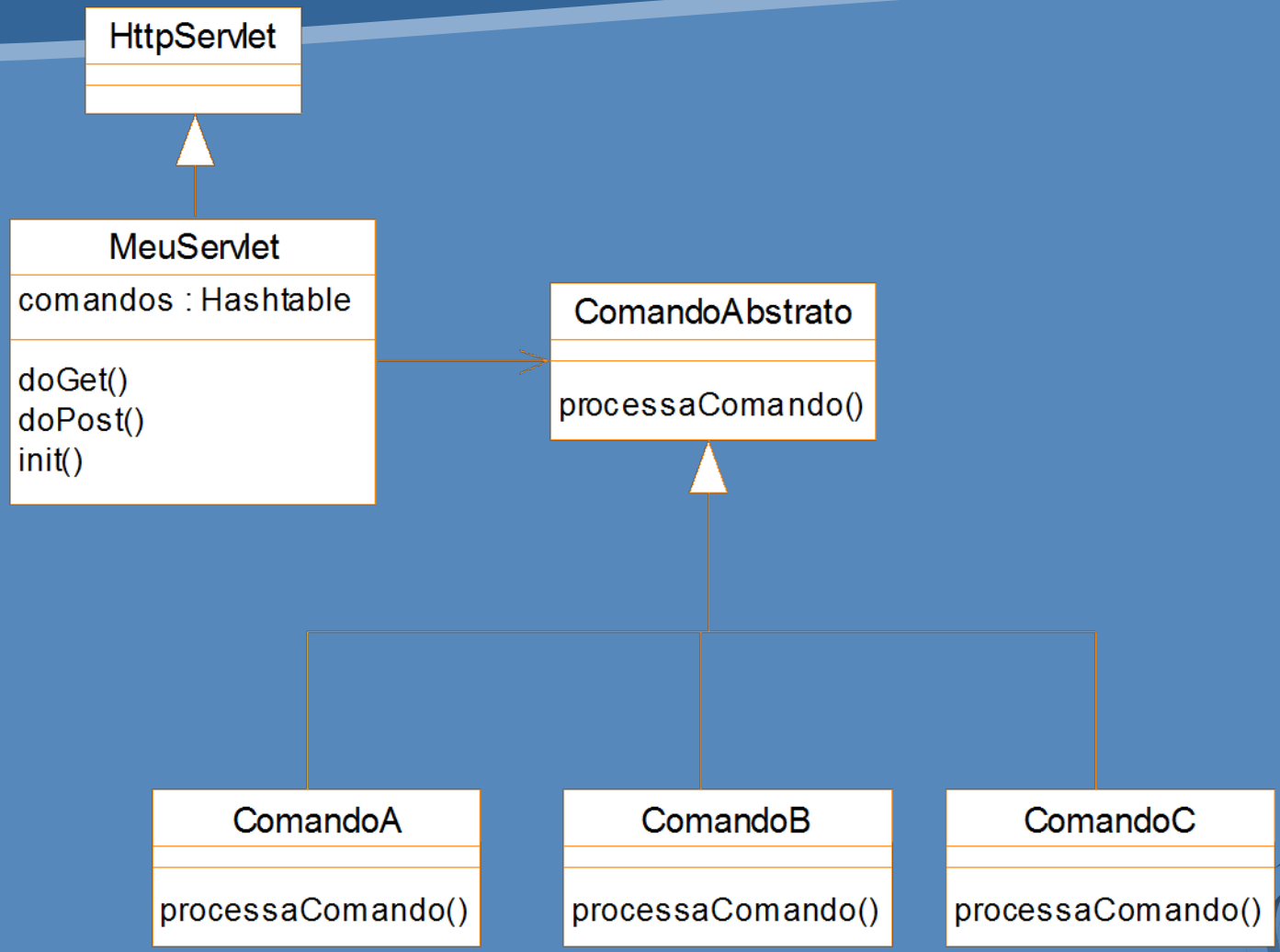
```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String comando = request.getParameter( "Comando" );
System.out.println( comando );
if ( comando.equalsIgnoreCase( "Login" ) )
{
   //...
}
if ( comando.equalsIgnoreCase( "Comprar" ) )
{
   //...
}
if ( comando.equalsIgnoreCase( "Fechar" ) )
{
   //...
}
if ( comando.equalsIgnoreCase( "Logout" ) )
{
   //....
}
```

- Atributo hidden + Comando abs

Padrão *Command* (II)

- Atributo hidden + Comando abstrato + Hashtable

Padrão *Command* (III)



Padrão *Command* (IV)

- O Comando Genérico:

```
public abstract class GenericCommand
{

    public abstract Vector execute( HttpServletRequest req ,
                                    HttpServletResponse res );

}
```

- HttpServletRequest
  - Passa o comando a ser instanciado (input do formulário)
- HttpServletResponse
  - Contém o PrintWriter que deverá receber as respostas do Comando

Padrão *Command* (V)

- O Servlet deve possuir um atributo HashTable com todos os Commands
- Ao ser iniciado, cada Comando deve ser instanciado e inserido na HashTable

```
public class MeuServlet extends HTTPServlet
{
    private Hashtable commands;
```

Padrão *Command* (VI)

```
public abstract void init(ServletConfig config) throws ServletException
{
    commands = new Hashtable();

    AbstractCommand command = new CommandA();
    commands.put( "A" , command );

    command = new CommandB();
    commands.put( "B" , command );

    command = new CommandC();
    commands.put( "C" , command );
}
```

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    String cmd = request.getParameter( "Comando" );
    AbstractCommand command = ( AbstractCommand ) commands.get( cmd );
    command.processarComando( request , response  );
}
```

Padrão *Command* (VII)

ormAcessarContaTag.html - Bloco de notas

Arquivo Editar Localizar Ajuda

```
<FORM ACTION= "/servlet/MeuServlet" METHOD = "POST" >
<CENTER>
Acesse sua Conta na Comunidade de Negócios iDeal!
<HR>
<BR>
<TABLE ALIGN = "CENTER" WIDTH = "100%" CELLSPACING="2" CELLPADDING= "2" >
<TR>
<FONT color="BLACK" face="Verdana" size="2">
<TD ALIGN = "RIGHT"> Login: </TD>
<TD > <INPUT TYPE = "TEXT" NAME = "Login" ALIGN = "LEFT" SIZE = "15"> </TD>
</TR>
<TR>
<TD ALIGN = "RIGHT"> Senha: </TD>
<TD > <INPUT TYPE = "PASSWORD" NAME = "Senha" ALIGN = "LEFT" SIZE = "15"> </TD>
</TR>
</TABLE>
<BR>
<HR>
<INPUT TYPE="SUBMIT" NAME = "BtnEntrar" VALUE = "Entrar !"> </TD>
<Input type="hidden" name="Comando" value="ProcessarLoginProt">
</FONT>
</FORM>
```

Dicas de Modelagem