Java Web

AULA 08 – MVC

1

Objetivos e Conceitos

- Objetivos:
  - Apresentar o conceito do padrão MVC. Apresentar um arquitetura de aplicação.
- Conceitos:
  - MVC. Front Controller.

2

Tópicos

- MVC
- Passando Informações da Servlet para a JSP
- Passando Lista de Informações da Servlet para a JSP
- Várias Ações, Uma Servlet
- Arquitetura da Aplicação

3

MVC

JSP + SERVLETS

4

MVC

Padrão de Projeto de Software

Surgiu em meados da década de 80: Aplicações desktop

Objetivos
- Diminuir acoplamento
- Separar lógica de apresentação da lógica de negócio
- Reuso
- Paralelização do desenvolvimento

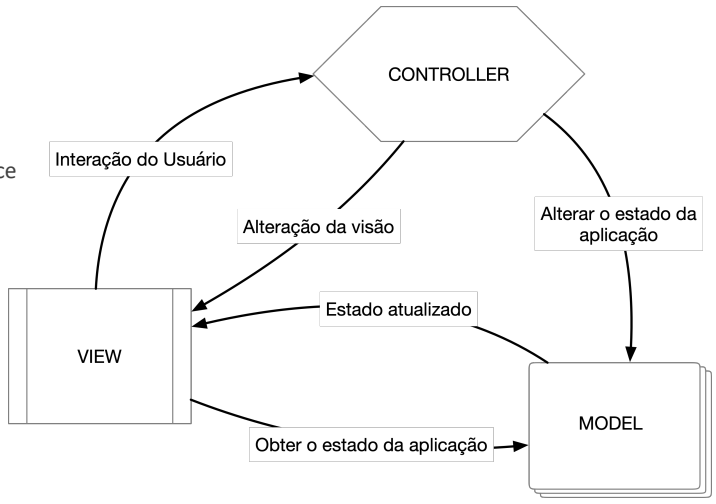
Surgiu com o Smalltalk
- Smalltalk-79 : Trygve Reenskaug introduz o MVC
- Smalltalk-80 : Implementação de uma biblioteca MVC

Artigo sobre MVC
- *Steve Burbeck. Applications programming in smalltalk-80: how to use model-view-controller (mvc).* 1987.
- https://www.researchgate.net/publication/238719652_Applications_programming_in_smalltalk-80_how_to_use_model-view-controller_mvc

5

MVC

Separa a aplicação em 3 partes
- ***View*** : Visão, apresentação dos dados para o usuário
- ***Model*** : Modelo, responsável pela lógica de negócio
- ***Controller*** : Controle, uma interface entre a Visão e o Modelo



Fonte: Adaptado de *Use a Cabeça! Padrões de Projetos, Elisabeth Freeman e Kathy Sierra.*

6

MVC.

Não confundir com a arquitetura 3-camadas (*3-tier*)

Arquitetura Cliente-Servidor / Corporativa / Possivelmente separados fisicamente

Camadas:

- Apresentação (*presentation*): o que é apresentado ao usuário
- Negócio (*logic/service*): lógica de negócio, acesso a dados, outros serviços
- Dados (*data*): armazenamento de dados

Em uma arquitetura 3-camadas, o MVC reside na camada de apresentação

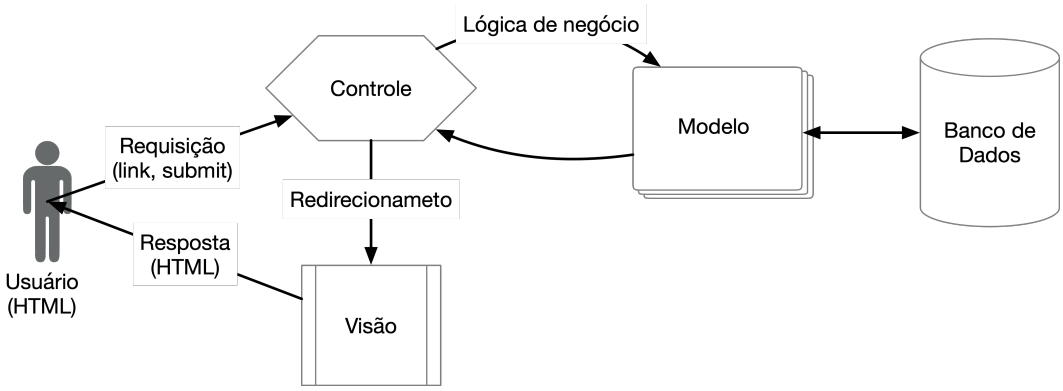
- Mesmo assim ainda tem-se as camadas de negócio e dados

7

MVC Model 2

Surge na Sun Microsystems

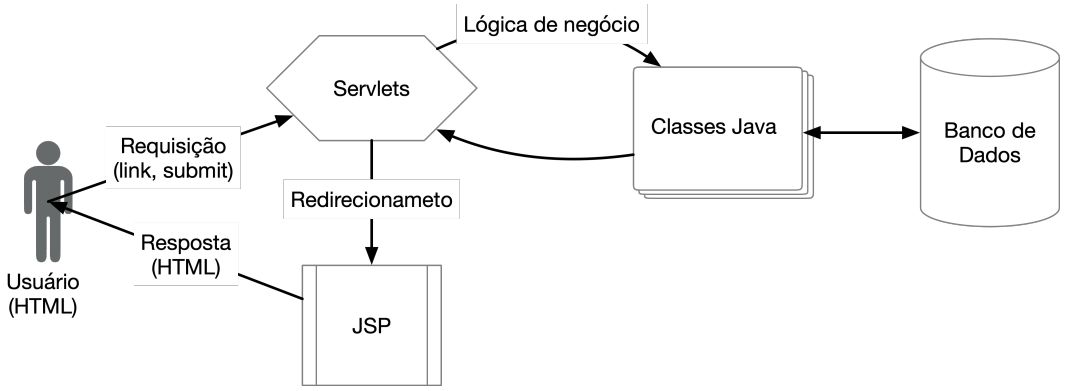
Aplicação de MVC em aplicações Java Web



8

MVC Model 2

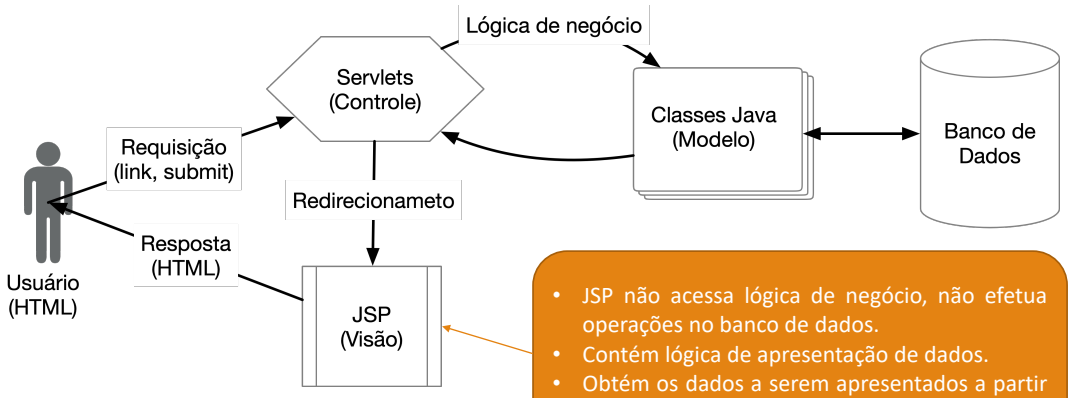
Adicionando as tecnologias disponíveis



9

MVC Model 2

Adicionando as tecnologias disponíveis

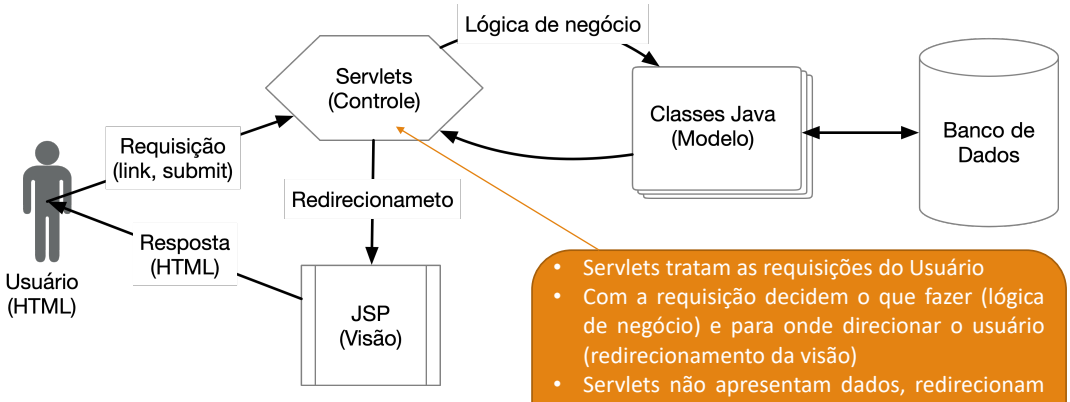


- JSP não acessa lógica de negócio, não efetua operações no banco de dados.
- Contém lógica de apresentação de dados.
- Obtém os dados a serem apresentados a partir do controle (Servlet, escopos, parâmetros)

10

MVC Model 2

Adicionando as tecnologias disponíveis

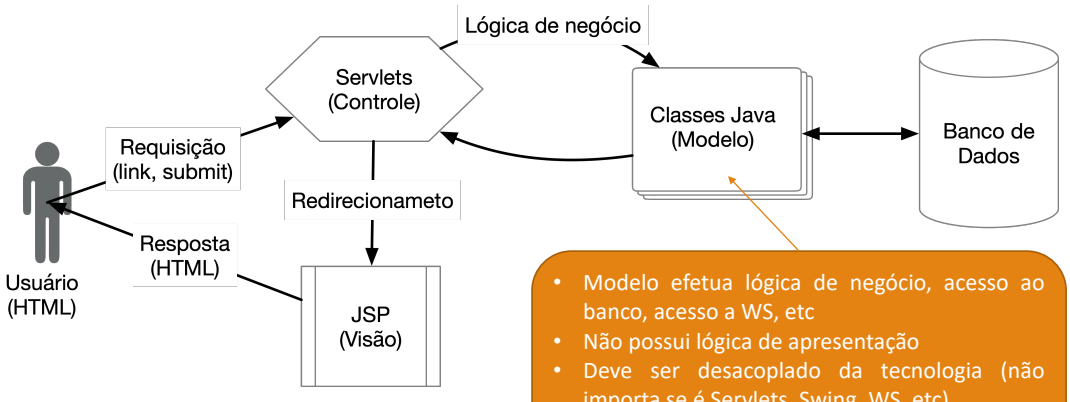


- Servlets tratam as requisições do Usuário
- Com a requisição decidem o que fazer (lógica de negócio) e para onde direcionar o usuário (redirecionamento da visão)
- Servlets não apresentam dados, redirecionam para uma JSP
- Servlets não executam lógica de negócio, invocam o Modelo

11

MVC Model 2

Adicionando as tecnologias disponíveis



- Modelo efetua lógica de negócio, acesso ao banco, acesso a WS, etc
- Não possui lógica de apresentação
- Deve ser desacoplado da tecnologia (não importa se é Servlets, Swing, WS, etc)
- Pode usar padrões de projetos, ex, Façade

12

MVC Model 2: Implementação.

JSP (Visão)
- Contém lógica de apresentação de dados, mostra informações, formulários, etc
- NÃO acessa o modelo, NÃO acessa DAOs, NÃO acessa banco de dados
- Os dados a serem apresentados vêm do Controle: são disponibilizados em algum escopo ou via parâmetro
- NÃO recebem submissões de formulários, as submissões são sempre no controle

Servlet (Controle)
- Recebe as requisições do usuário, os *submits* acontecem sempre em Servlets
- NÃO apresenta informações. Se precisar mostrar algo, redireciona para JSP (ex, erro)
- Dada uma requisição:
  - Decide o que fazer no Modelo
  - Disponibiliza informações para a Visão
  - Decide para qual tela (JSP) deve redirecionar
- Redireciona para uma JSP

Classes Java (Modelo)
- Efetua lógica de negócio, acessa banco de dados, acessa WS
- É desacoplado da tecnologia: NÃO recebe request/response, NÃO manipula sessão

13

Controladora

Todas as controladoras executam (+/-) 4 passos:

1. **PASSO 1:** Obter parâmetros
2. **PASSO 2:** Processar/Validar/Converter Parâmetros
3. **PASSO 3:** Efetuar Ação
4. **PASSO 4:** Redirecionamento (Passando ou não parâmetros)

14

Controladora.

```
@WebServlet(urlPatterns = {"/Controladora"})
public class Controladora extends HttpServlet {
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                   throws ServletException, IOException {

      // PASSO 1: Obter Parâmetros
      String strId = request.getParameter("id");

      // PASSO 2: Processar/Validar/Converter Parâmetros
      int id = Integer.parseInt(strId);

      // PASSO 3: Efetuar Ação
      ClientesDAO dao = new ClientesDAO();
      dao.remover (id);

      // PASSO 4: Redirecionamento
      RequestDispatcher rd = getServletContext().getRequestDispatcher("/mostrar.jsp");
      rd.forward(request, response);
   }
}
```


15

Exemplo

```
<html><head><title>index.jsp</title></head>
<body>
<form action="Controladora" method="post" >
   Nome:   <input type="text" name="nome" /><br/>
   E-mail: <input type="text" name="email" /><br/>
   <input type="submit" value="Salvar" />
   <input type="reset" value="Limpar" />
</form>
</body>
</html>
```


16

Exemplo

```
...
public class Controladora extends HttpServlet {
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                   throws ServletException, IOException {

      String nm = request.getParameter("nome");
      String em = request.getParameter("email");

      // faz o que tem que fazer, INVOCA O MODELO

      RequestDispatcher rd = getServletContext().
               getRequestDispatcher("/mostrar.jsp");
      rd.forward(request, response);
   }
}
```


17

Exemplo.

```
<html><head><title>mostrar.jsp</title></head>
<body>
    <h1>Ação efetuada com sucesso</h1>
</body>
</html>
```


18

Exercícios

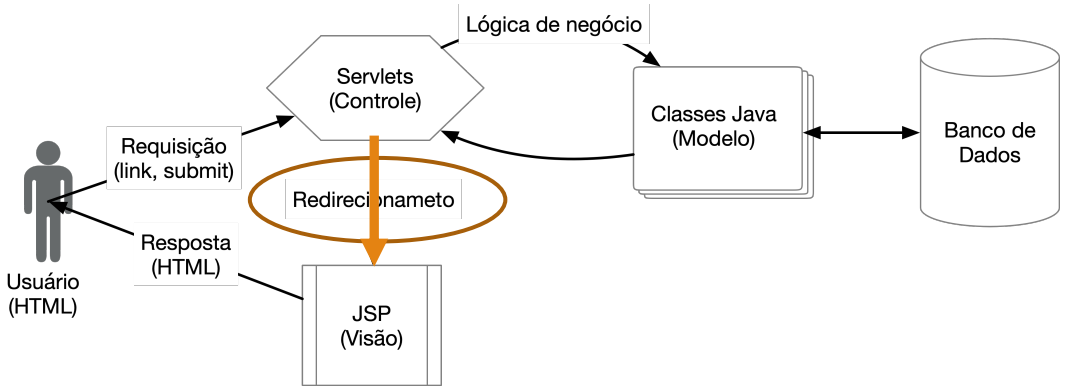
1. Executar a aplicação anterior

19

Passando Informações da Servlet para a JSP

20

Implementação



21

Implementação

Na Servlet (Controladora), usa-se o *request* para passar um atributo no Escopo da Requisição

```
request.setAttribute("mensagem", "oi");
RequestDispatcher rd = request.getRequestDispatcher("tela.jsp");
rd.forward(request, response);
```

Na JSP destino (Visão - **tela.jsp**), usa-se para obter o atributo:

```
<% String msg = (String)request.getAttribute("mensagem"); %>
<h2><%= msg %></h2>
```

OU, usando EL (*Expression Language*)

```
<h2>${mensagem}</h2>
```


22

Exemplo

```
<html><head><title>index.jsp</title></head>
<body>
<form action="Controladora" method="post" >
   Nome:   <input type="text" name="nome" /><br/>
   E-mail: <input type="text" name="email" /><br/>
   <input type="submit" value="Salvar" />
   <input type="reset" value="Limpar" />
</form>
</body>
</html>
```


23

Exemplo

```
...
@WebServlet(urlPatterns = {"/Controladora"})
public class Controladora extends HttpServlet {
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                   throws ServletException, IOException {

      String nm = request.getParameter("nome");
      String em = request.getParameter("email");

      // faz o que tem que fazer (ex, banco de dados)

      request.setAttribute("mensagem", "Dados corretos");
      RequestDispatcher rd = getServletContext().
                             getRequestDispatcher("/mostrar.jsp");
      rd.forward(request, response);
   }
}
```


24

Exemplo

```
<html><head><title>mostrar.jsp</title></head>
<body>
  <h1>Ação efetuada com sucesso</h1>
  Mensagem: <%= (String)request.getAttribute("mensagem") %>
</body>
</html>
```


25

Exemplo

```
<html><head><title>mostrar.jsp</title></head>
<body>
  <h1>Ação efetuada com sucesso</h1>
  Mensagem: ${mensagem}
</body>
</html>
```


26

Exercícios

1. Criar a aplicação anterior com as seguintes variações:
   a. Passar o parâmetro no escopo da requisição (como no exemplo)
   b. Passar o parâmetro na URL (alterar a Servlet e o JSP)
   c. Passar dois parâmetros, um no escopo da requisição e um na URL. Alterar a Servlet e o JSP)

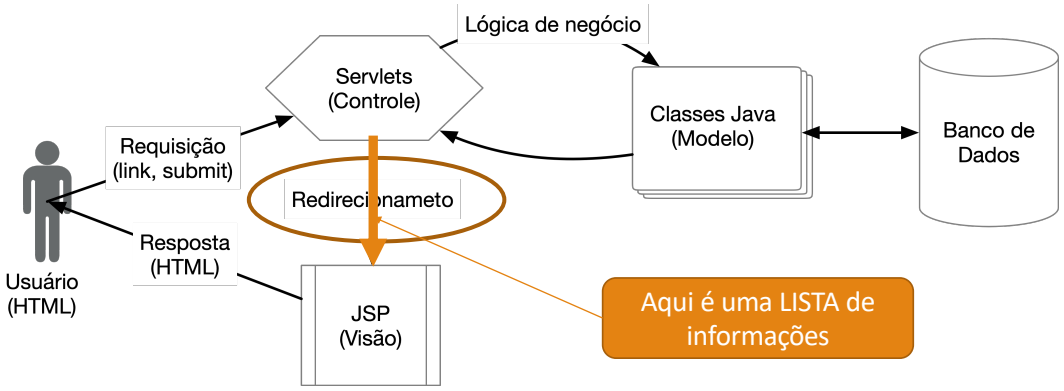
- Na Servlet não é necessário efetuar nenhuma ação, basta o redirecionamento com parâmetro(s)
- Testar o comando em EL para mostrar o(s) dado(s) na JSP

27

Passando Lista de Informações da Servlet para a JSP

28

Implementação



29

Implementação

A diferença desta para o anterior é o tipo do dado que está sendo passado

Será passado uma Lista de Strings

```
List<String> arr = new ArrayList<String>();
arr.add("Cebola");
arr.add("Batata");
request.setAttribute("dados", arr);
RequestDispatcher rd = request.getRequestDispatcher("lista.jsp");
rd.forward(request, response);
```

Na JSP destino (**`lista.jsp`**), usa-se para obter o atributo:

```
<% List<String> lst = (List<String>)request.getAttribute("dados"); %>
```

Para mostrar a lista, sem *scriptlet*, é necessário um laço em JSTL

30

Exemplo

```
<html><head><title>index.jsp</title></head>
<body>
<form action="Controladora" method="post" >
   Text:   <input type="text" name="texto" /><br/>
   <input type="submit" value="Pesquisar" />
   <input type="reset" value="Limpar" />
</form>
</body>
</html>
```


31

Exemplo

```
…
@WebServlet(urlPatterns = {"/Processa"})
public class Controladora extends HttpServlet {
   protected void doPost(HttpServletRequest request,
                         HttpServletResponse response)
                   throws ServletException, IOException {

      String texto = request.getParameter("texto");

      // faz o que tem que fazer

      List<String> lista = new ArrayList<String>();
      lista.add("Cebola");
      lista.add("Batata");
      request.setAttribute("dados", lista);
      RequestDispatcher rd = request.getRequestDispatcher("/mostrar.jsp");
      rd.forward(request, response);
   }
}
```


32

Exemplo com Scriptlet

```
<html><head><title>mostrar.jsp</title></head>
<body>
  <h1>Ação efetuada com sucesso</h1>
  Dados: <br/>
  <%
    List<String> arr = (List<String>)request.getAttribute("dados");
    for (String x: arr) {
        out.println(x + "<br/>");
    }
  %>
</body>
</html>
```


33

Exemplo com JSTL

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><head><title>mostrar.jsp</title></head>
<body>
  <h1>Ação efetuada com sucesso</h1>
  Dados: <br/>
  <c:forEach var="x" items="${dados}" >
     ${x} <br/>
  </c:forEach>
</body>
</html>
```

Adicionar a biblioteca JSTL no projeto

34

Exercícios

1. Criar a aplicação anterior, mostrando os dados das duas formas
   a. Com Scriptlet
   b. Com JSTL

35

Várias Ações, Uma Servlet

36

Várias Ações

Pode ser necessário que uma Servlet atenda a requisições de diferentes telas

Exemplo:

- Tela Inserir Pessoa
- Tela Pesquisar Pessoa

Pode ficar inviável fazer uma servlet para cada ação

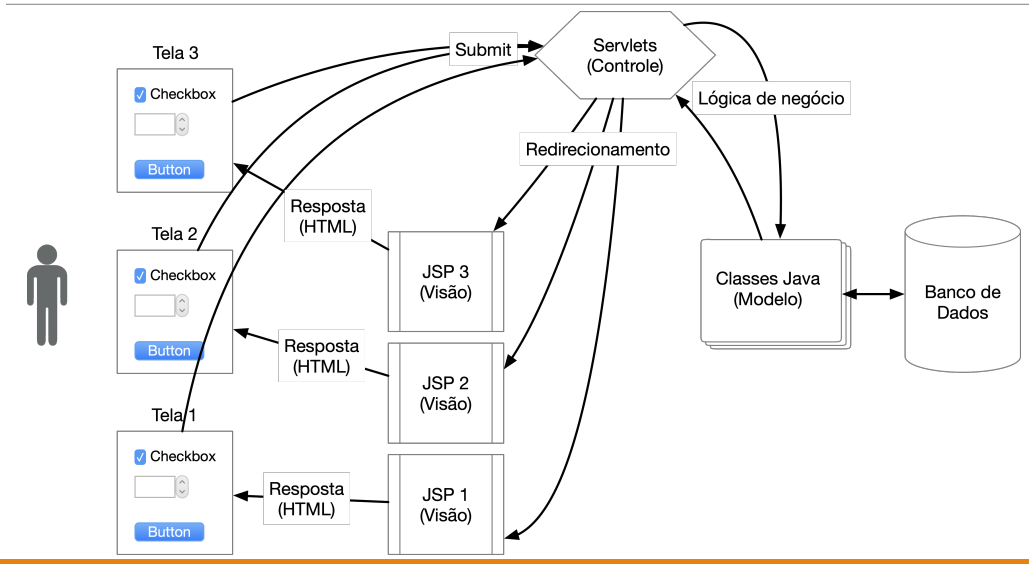
Passa-se um parâmetro (ex. **`action`**) para a Servlet, indicando qual é a ação a ser executada

Na Servlet, obtém-se este parâmetro e faz-se um **`if`** para cada ação pertinente

Cada condição do **`if`** efetua uma operação e redireciona para um local diferente

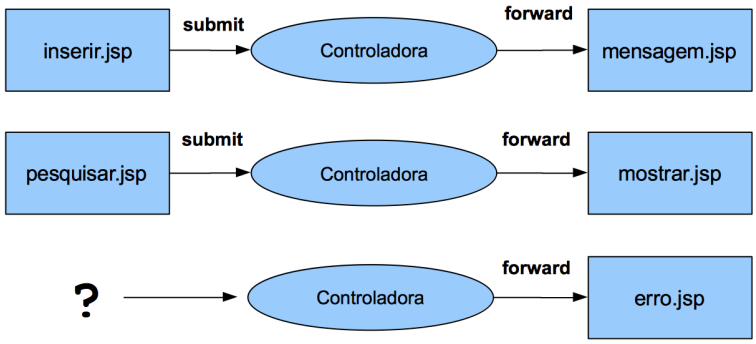
37

Várias Ações



38

Várias Ações.



39

Exemplo

```
<html><head><title>inserir.jsp</title></head>
<body>
<form action="Controladora?action=inserir" method="post" >
   Nome:   <input type="text" name="nome" /><br/>
   E-mail: <input type="text" name="email" /><br/>
   <input type="submit" value="Inserir" />
   <input type="reset" value="Limpar" />
</form>
</body>
</html>
```


40

Exemplo

```
<html><head><title>pesquisar.jsp</title></head>
<body>
<form action="Controladora?action=pesquisar" method="post" >
   Texto:   <input type="text" name="texto" /><br/>
   <input type="submit" value="Pesquisar" />
   <input type="reset" value="Limpar" />
</form>
</body>
</html>
```


41

Exemplo

```
@WebServlet(urlPatterns = {"/Controladora"})
public class Controladora extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
    String action = request.getParameter("action");
    if (action.equals("inserir")) {
       // insere
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/mensagem.jsp");
       rd.forward(request, response);
    }
    else if (action.equals("pesquisar")) {
       // pesquisa
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/mostrar.jsp");
       rd.forward(request, response);
    }
    else {
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/erro.jsp");
       rd.forward(request, response);
    }
  }
}
```


42

Exemplo

```
@WebServlet(urlPatterns = {"/Controladora"})
public class Controladora extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {
    String action = request.getParameter("action");
    if (action.equals("inserir")) {
       // insere
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/mensagem.jsp");
       rd.forward(request, response);
    }
    else if (action.equals("pesquisar")) {
       // pesquisa
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/mostrar.jsp");
       rd.forward(request, response);
    }
    else {
       RequestDispatcher rd = getServletContext().getRequestDispatcher("/erro.jsp");
       rd.forward(request, response);
    }
  }
}
```

PENSE: Qual o problema neste código??

43

Exemplo

```
<html><head><title>mensagem.jsp</title></head>
<body>
    <h1>Usuário inserido com sucesso</h1>
</body>
</html>
```


44

Exemplo

```
<html><head><title>mostrar.jsp</title></head>
<body>
    <h1>Dados para serem mostrado</h1>
</body>
</html>
```


45

Exemplo.

```
<html><head><title>erro.jsp</title></head>
<body>
    <h1>Ação não reconhecida</h1>
</body>
</html>
```


46

Quantas Servlets Implementar?.

1 servlet por ação

- Facilmente o número de arquivos/Servlets fica impraticável

1 servlet para a aplicação toda

- Padrão de Projeto ***FrontController***
- Bom para *Frameworks*
- Problemas que podem aparecer:
- Ou a complexidade da Servlet aumenta muito (configuração, xml, generalismo, etc)
- Ou seu tamanho (quantidade de condicionais) aumenta muito

Equilíbrio: 1 Servlet por módulo, por assunto, etc

- Ex. Uma Servlet para as ações de Pessoa, uma para Atendimento, uma para funcionário, etc.

47

Exercícios

1. Criar um arquivo **`index.html`** que contém dois links, um para **`inserir.jsp`** e outro para **`mostrar.jsp`**
   - Criar a aplicação anterior, com as várias ações
   - Arrumar o erro no código apontado no slide

48

Arquitetura da Aplicação

49

Arquitetura

Usar MVC

- **Visão** : JSP
  - Gera a saída para o usuário
- **Controle** : Servlet
  - Recebe requisições do usuário
  - Invoca o Modelo para efetuar a operação de negócio
  - Trata o retorno e erros
  - Redireciona para a Visão
- **Modelo** : Classes Java
  - DAOs, Classes de Domínio (Java Beans), outras classes
  - Efetuam uma operação de negócio. Ex. Inserir uma Pessoa
  - Deve ser totalmente desacoplada/inconsciente da tecnologia

50

Arquitetura

Para separar Visão e Controle do Modelo, usa-se uma classe intermediária
- Padrão de Projeto FAÇADE

É um Padrão de Projeto do GoF (*Gang of Four*)
- *Design Patterns: Elements of Reusable Object-Oriented Software (1994)*
- "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Usado para "esconder" a complexidade de um subsistema ou parte do sistema

Provê uma interface simplificada ao subsistema

51

Arquitetura

No nosso caso, vamos adicioná-lo para esconder a complexidade das operações de negócio

Objetivo
- Esconder a complexidade do acesso ao banco de dados
- Esconder tratamentos de erro específicos, instanciações
- Desacoplar com a tecnologia: não usar *request*/*response*

Cada método do *Façade* implementa uma ação de negócio solicitada por uma requisição

52

Arquitetura

Classe Façade

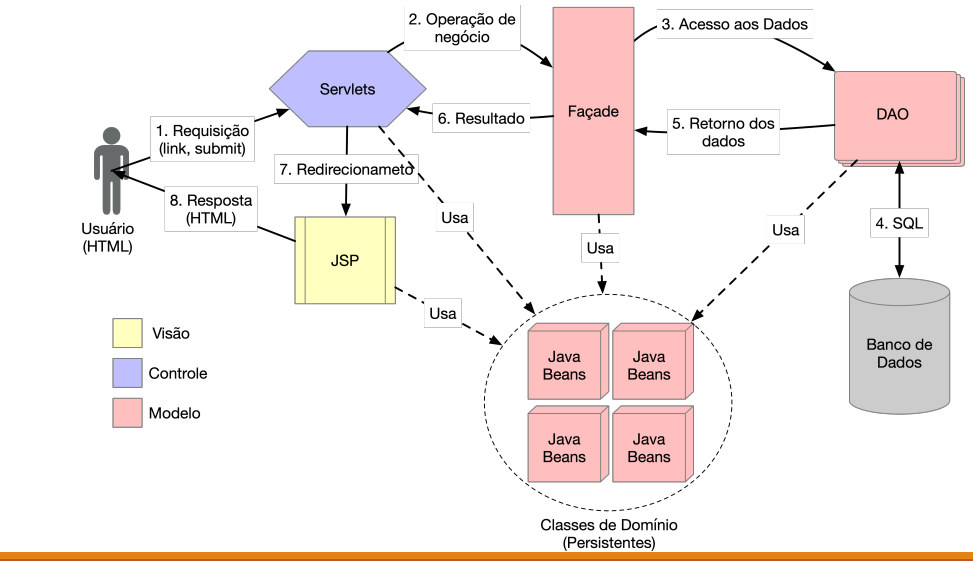
- Classe POJO
- Somente métodos de negócio
- Não recebe objetos de tecnologia (*request*/*response*), somente Java Beans
- Métodos podem ser estáticos ou não:

```
ClientesFacade.inserirCliente(c);
```

```
ClientesFacade facade = new ClientesFacade();
facade.inserirCliente(c);
```


53

Arquitetura.



54

Façade

```
package com.ufpr.tads.web2.facade;

public class ClientesFacade {
   public static void inserirCliente(Cliente c)
                                throws InserirClienteException {
      ...
   }
   public static void removerCliente(int id)
                                throws RemoverClienteException {
      ...
   }
   ...
}
```


55

Façade: Uso na Servlet.

```
@WebServlet(urlPatterns = {"/Controladora"})
public class Controladora extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

      // PASSO 1: Obter Parâmetros
      String strId = request.getParameter("id");

      // PASSO 2: Processar/Validar/Converter Parâmetros
      int id = Integer.parseInt(strId);

      // PASSO 3: Efetuar Ação (Usando Façade)
      ClientesFacade.removerCliente(id);

      // PASSO 4: Redirecionamento
      RequestDispatcher rd = getServletContext().getRequestDispatcher("/mostrar.jsp");
      rd.forward(request, response);
   }
}
```


56

Tratamento de Erros

Tem-se basicamente 3 pontos de erros no código anterior

1. Parâmetro "id" não passado

```
String strId = request.getParameter("id");
```

Solução:

```
String strId = request.getParameter("id");
if (strId == null) {
   request.setAttribute("mensagem", "Invocação inválida: id é nulo");
   RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
   rd.forward(request, response);
   return;
}
```


57

Tratamento de Erros

2. Foi passado um valor que não pode ser convertido para inteiro

```
int id = Integer.parseInt(strId);
```

Solução:

```
try {
    int id = Integer.parseInt(strId);
    .........
}
catch(NumberFormatException e) {
  request.setAttribute("mensagem", "Id para remoção inválido: " +
                                    strId);
  RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
  rd.forward(request, response);
}
```


58

Tratamento de Erros

3. Erro na remoção do Cliente

```
ClientesFacade.removerCliente(id);
```

Solução:

```
try {
    .........
    ClientesFacade.removerCliente(id);
    .........
}
catch(RemoverClienteException e) {
  request.setAttribute("mensagem", "ERRO: " + e.getMessage());
  RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
  rd.forward(request, response);
}
```


59

Tratamento de Erros

NUNCA deixar erros sem tratamento:

```
try {
      ...
}
catch(SQLException e) {
}
```

Criar uma hierarquia de Exceções para tratar os erros da aplicação

Não propagar erros de Banco (ex. SQLException)

Desacoplar a tecnologia de **Apresentação** da tecnologia de **Armazenamento**

Tratar erros de infra, na infra

Ao criar o erro da aplicação, adicionar a "causa"

60

Tratamento de Erros

```
public class RemoverClienteException extends Exception {
    public RemoverClienteException() {
    }
    public RemoverClienteException(String string) {
        super(string);
    }
    public RemoverClienteException(String string,
                                   Throwable thrwbl) {
        super(string, thrwbl);
    }
}
```


61

Tratamento de Erros

```
public class ClientesFacade {
   public static void removerCliente(int id)
                                throws RemoverClienteException {
      try {
            // Código que gera um SQLException
      }
      catch(SQLException e) {
          throw new RemoverClienteException(
                          "Erro removendo cliente: id=" + id, e);
      }
   }
}
```


62

Tratamento de Erros.

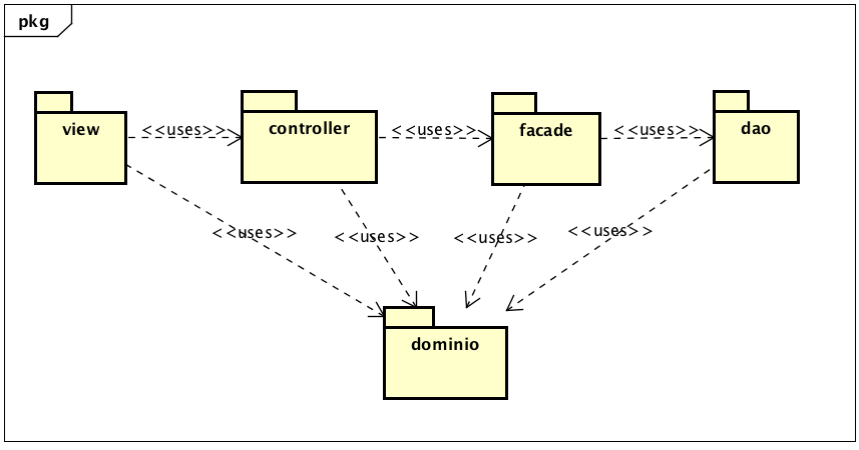
```
String strId = request.getParameter("id");
if (strId == null) {
   request.setAttribute("mensagem", "Invocação inválida: id é nulo");
   RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
   rd.forward(request, response);
   return;
}
try {
   int id = Integer.parseInt(strId);

   ClientesFacade.removerCliente(id);

   RequestDispatcher rd = request.getRequestDispatcher("/mostrar.jsp");
   rd.forward(request, response);
}
catch(NumberFormatException e) {
   request.setAttribute("mensagem", "Id para remoção inválido: " + strId);
   RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
   rd.forward(request, response);
}
catch(RemoverClienteException e) {
   request.setAttribute("mensagem", "ERRO: " + e.getMessage());
   RequestDispatcher rd = request.getRequestDispatcher("/erro.jsp");
   rd.forward(request, response);
}
```


63

Diagrama de Classes de Implementação



64

Diagrama de Classes de Implementação.

Para cada pacote, fazer seu diagrama de classes

- **`view`**, **`controller`**, **`facade`** e **`dao`**:
  - Não possuem associações entre si
  - Adicionar as classes e apresentar o diagrama
  - Não há necessidade de mostrar atributos e métodos da plataforma, a não ser que sejam invocados nos Diagramas de Sequência
- **`dominio`**:
  - Aqui estão as classes de negócio
  - Estas contém uma modelagem rica, com associações, herança, interface, polimorfismo, etc
  - Este é o diagrama de classes de análise, ou diagrama de classes de modelo, etc

65

Exercícios..

1. Implementar:
   a. **`index.html`**: formulário com um campo id que submete para `ProcessaServlet`
   b. **`ProcessaServlet`**: implementa o controlador usando *façade* com os tratamentos de erros
   c. **`ClientesFacade`**: implementar o método `removerCliente()` conforme apresentado, simulando uma chamada no banco. Ex: se receber 1, dá certo; se receber 2, levanta exceção
   d. **`RemoverClienteException`**: exceção que ocorre quando a remoção dá erro
   e. **`mostrar.jsp`**: mostra uma mensagem de usuário removido com sucesso
   f. **`erro.jsp`**: mostra uma mensagem de erro recebida no escopo da requisição, em vermelho

66