MC0037 – Programação para Web

Aula 5: MVC: Model-View-Controller

2 Q / 2015

Aula de hoje: Roteiro

- Padrão MVC: Model-View-Controller

Separação das responsabilidades

- Um sistema de fácil manuntenção é aquele em que as responsabilidades estão separadas e implementadas em locais bem-definidos
- Exemplos de responsabilidades:
  - Salvar um dado (ex.: aluno) no banco de dados
  - Decisão se um dado (ex.: aluno) pode ser removido ou não do banco de dados
  - Montagem da página de saída HTML para exibição dos dados

- Idealmente, essas responsabilidades devem estar separadas em camadas. Podemos distinguir 3 camadas:
  - Camada de apresentação
  - Camada das regras de negócio
  - Camada de acesso aos dados (ou camada de persistência)

MVC

- MVC (Model-View-Controller): é um padrão de arquitetura de software que visa a separar a lógica de negócio da lógica de apresentação

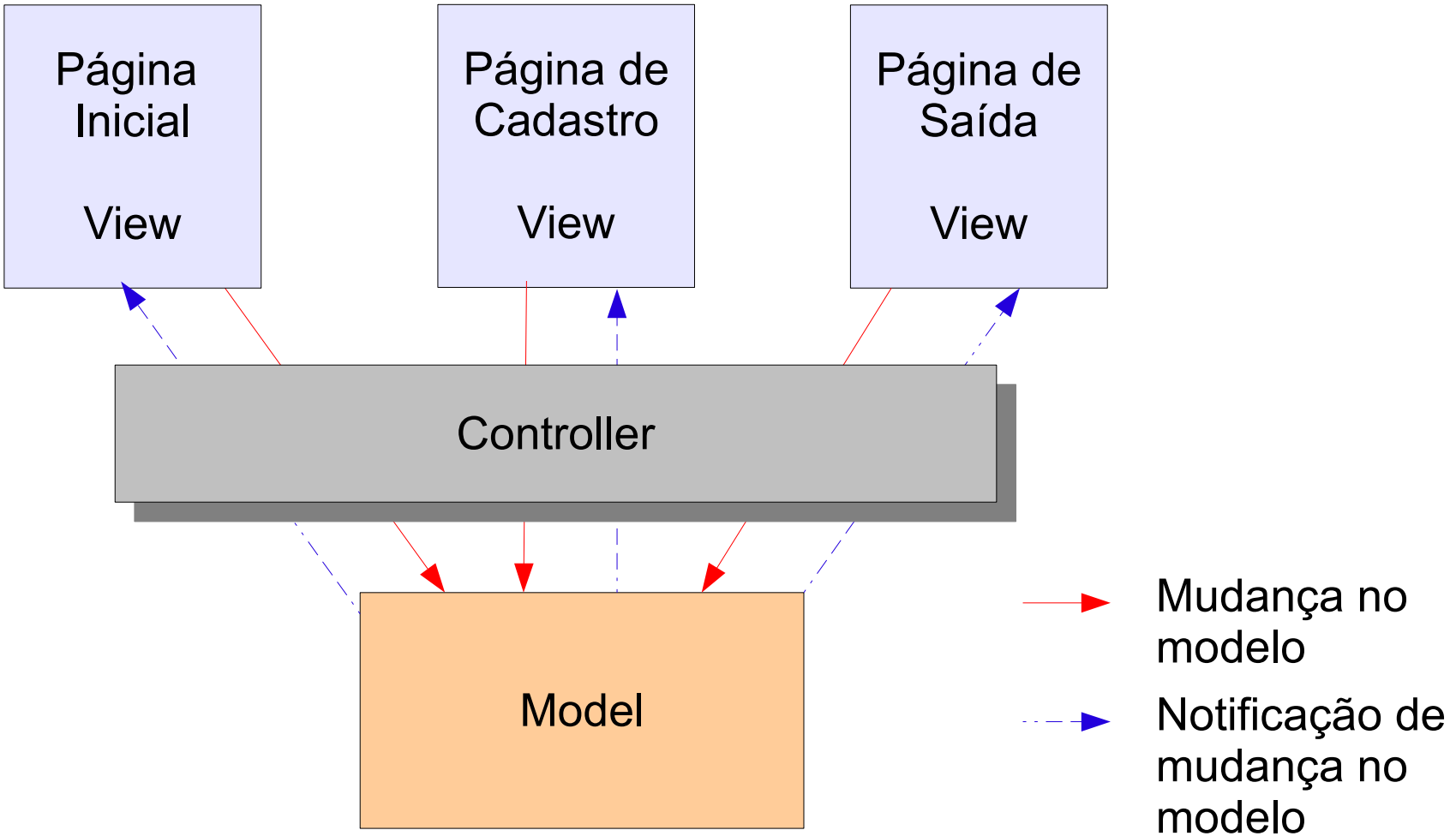
- Model (Modelo): acesso aos dados, contém as regras de negócio (responsável por manter o estado da aplicação)

- View (Visão): responsável pela apresentação (interfaces), o que é apresentado para o usuário

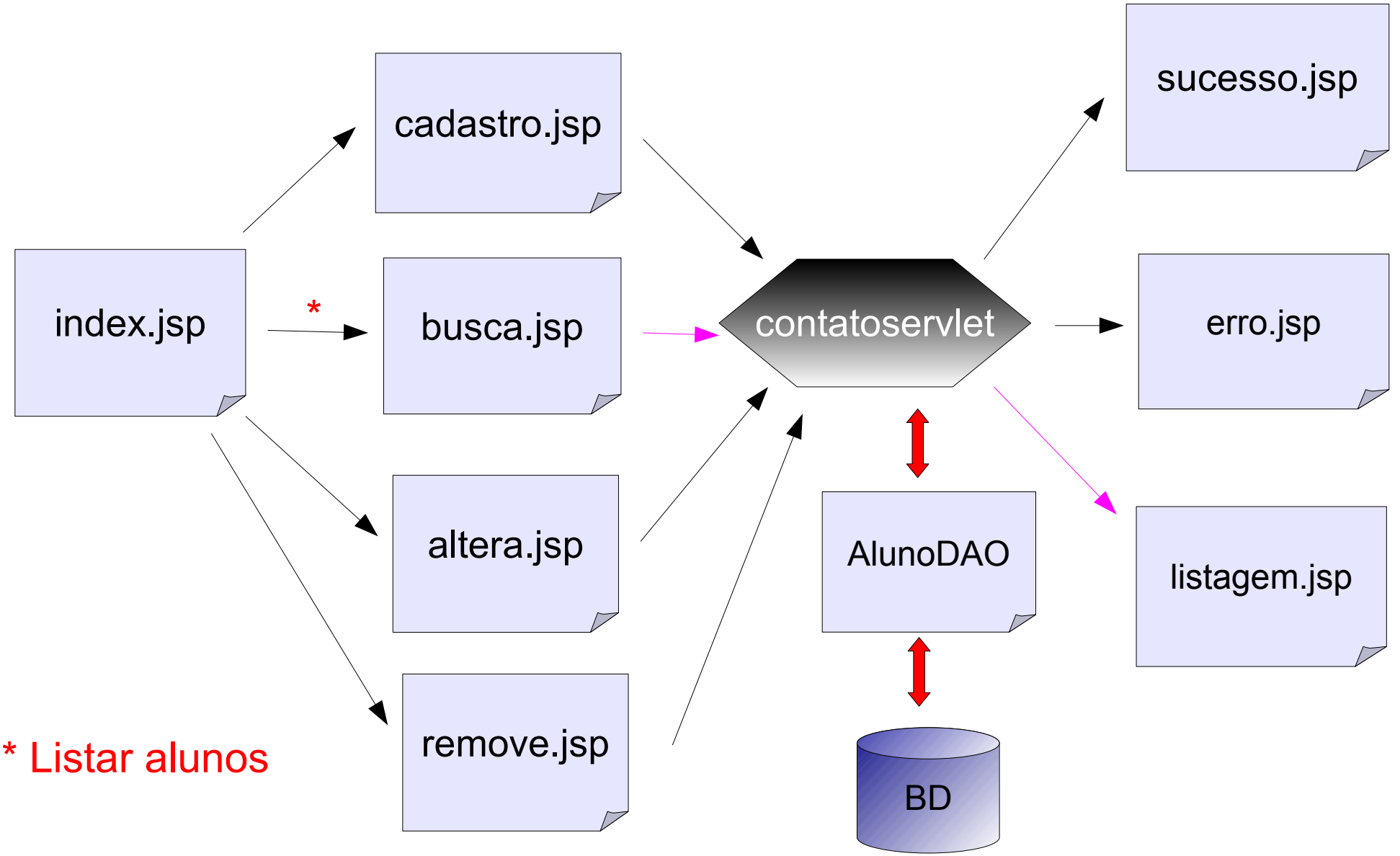
- Controller (Controlador): faz a intermediação entre Modelo e Visão, gerencia as interações com o usuário (através da view) e dispara atualizações para o modelo

MVC

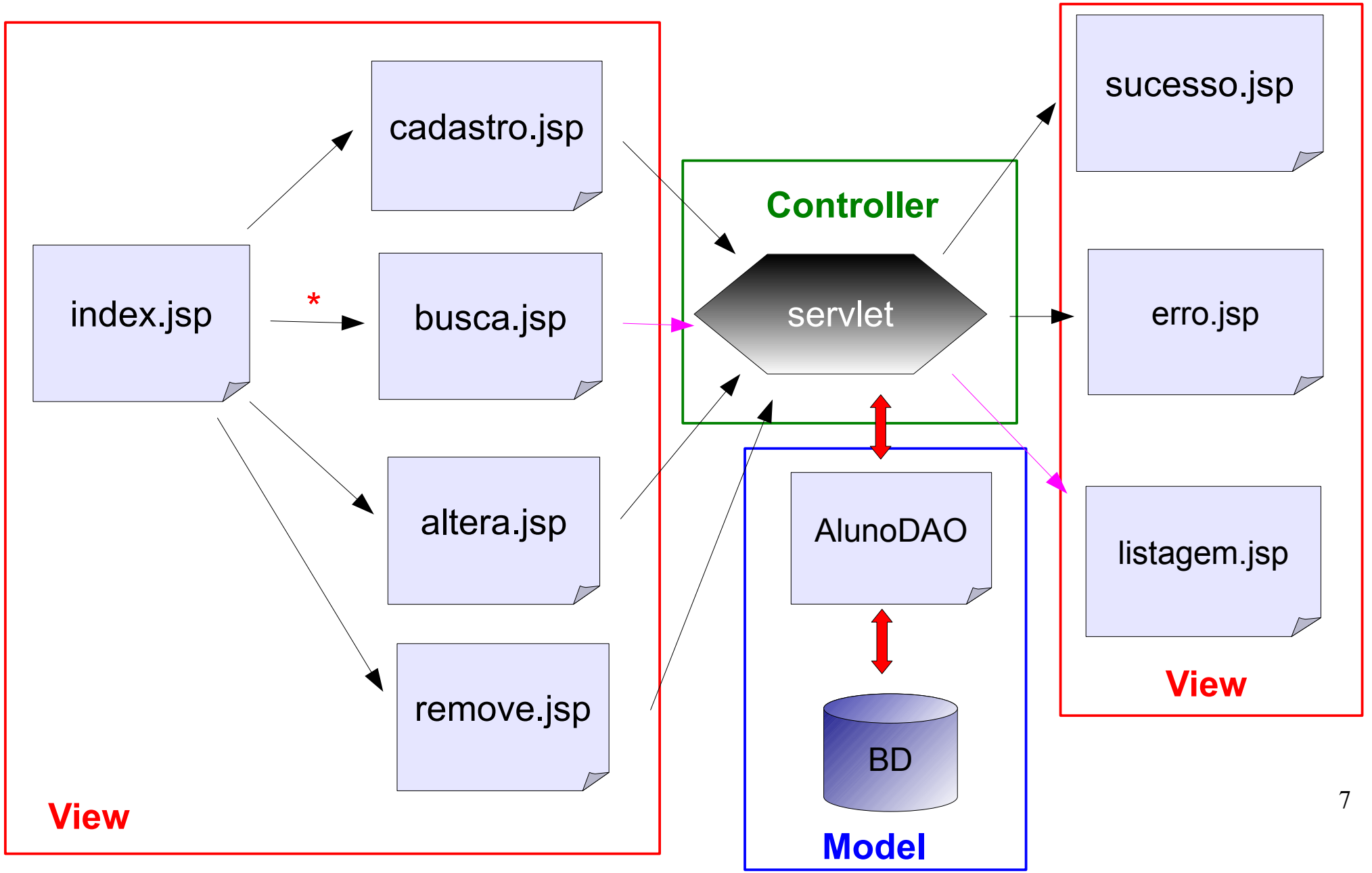
- Interação entre View, Controller e Model



Exemplo: fluxo de navegação das páginas



MVC: separação de responsabilidades



Transferindo dados com JavaBeans

- Separando as regras de negócio da apresentação, precisamos de alguma forma para poder transferir os dados entre eles

- De maneira simplificada, as informações do usuário são recebidas através da página e estas informações precisam ser transferidas aos objetos que fazem algum processamento com o conteúdo

- Para isso, são utilizados os JavaBeans, que são objetos que encapsulam conjuntos relacionados de dados para transferência entre diferentes componentes da aplicação

- Relembrando:
- Um JavaBean é uma classe Java com um construtor vazio e um conjunto de propriedades (além de métodos get / set para cada atributo da classe)

DAO

- DAO (Data Access Object) é um padrão de projeto do Java EE patterns
- A idéia básica do DAO é encapsular as operações básicas de acesso ao BD, conhecidas como CRUD (Create, Read, Update, Delete)
- Na prática, são classes que tratam exclusivamente das operações do banco de dados

Vantagens do MVC para aplicações web

- Simplicidade
  - Cada componente pode ser projetado e implementado separadamente, reduzindo a complexidade do problema
- Independência
  - Os componentes podem ser trocados ou modificados conforme necessário (ex.: uma interface (página inicial) pode ser atualizada ou trocada sem afetar outros componentes, pode-se ter uma interface para diferentes dispositivos, etc.)
- Escalabilidade
- Facilidade de manutenção

Aplicando o padrão MVC

- Ao aplicar o padrão MVC em uma aplicação web Java, podemos ter o seguintes módulos:
- Componente View
  - HTML, JSP, imagens, scripts, taglibs, etc., ou seja, componentes com a finalidade de gerar a apresentação (interface) do usuário
- Componente Controller
  - Servlets que respondem às requisições HTTP
  - O controller trata cada ação originada da view ativando o componente apropriado do modelo para tratar a requisição
  - Seleciona a view de resposta para a ação
- Componente Model
  - JavaBeans e outras classes *helper* que constituem a parte da aplicação responsável em manter e manipular o estado da aplicação (lógica da aplicação)
  - Inclui também o banco de dados

Exemplo: aplicando o padrão MVC

- Reduzindo o número de servlets: agrupando as operações em um único servlet: ControllerServlet

```
@WebServlet("/controller")
public class ControllerServlet extends HttpServlet {
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
   String opcao = req.getParameter("opcao");
   // obtem parametros do request
   String nome = req.getParameter("nome");
   String email = req.getParameter("email");
   String endereco = req.getParameter("endereco");
   // instancia objeto Aluno e seta os atributos
   Aluno aluno = new Aluno();
   aluno.setNome(nome);
   aluno.setEmail(email);
   aluno.setEndereco(endereco);
   AlunoDAO dao = new AlunoDAO();
   // continua ...
```

Exemplo: aplicando o padrão MVC

- Reduzindo o número de servlets: agrupando as operações em um único servlet

```
    // continuacao
    if (opcao.equals("inserir")) {
        dao.insere(aluno);
        req.setAttribute("msg", "Aluno: " + aluno.getNome() + "
cadastrado com sucesso!");
        RequestDispatcher rd = req.getRequestDispatcher("/sucesso.jsp");
        rd.forward(req, resp);
    }
    else if (opcao.equals("alterar")) {
        // ...
    }
    else if (opcao.equals("remover")) {
        // ...
    }
  }
}
```

Anexando um atributo (msg) no escopo da requisição.

Redirecionando a saída para a página sucesso.jsp.

Exemplo: aplicando o padrão MVC

- insere.jsp: inserindo um campo "*hidden*" para passar o tipo de operação

```
<body>
    <h3>Inserção de Aluno:</h3>
    <form action="controller" method="post">
            Nome: <input type="text" name="nome" /><br>
            Email: <input type="text" name="email" /><br>
            Endereço: <input type="text" name="endereco" /><br><br>
            <input type="submit" value="Gravar" />
            <input type="hidden" name="opcao" value="inserir" />
    </form>
</body>
```

Exemplo: aplicando o padrão MVC

- sucesso.jsp: página para apresentar a mensagem de saída

```
<body>
    <h3>Mensagem:</h3>
    ${requestScope.msg}
    <br><br>
    <a href="index.jsp">Voltar</a>
</body>
```

Obtendo o atributo (msg) do escopo da requisição.

Separando a lógica de negócio do Servlet

- Para tornar o nosso projeto (ProgradWeb) completamente MVC, é preciso separar a lógica de negócio (operações insere, altera, remove, que fazem parte do modelo) do servlet
- Vamos utilizar polimorfismo: criando uma interface Acao:

```
public interface Acao {
  public void executa(HttpServletRequest req,
      HttpServletResponse resp) throws Exception;
}
```

Separando a lógica de negócio do Servlet

- As classes que tratam as operações insere, altera, remove devem implementar a interface Acao:

```
public class InsereAluno implements Acao {
   public void executa(HttpServletRequest req, HttpServletResponse resp)
       throws Exception {
       // obtem parametros do request
       String nome = req.getParameter("nome");
       String email = req.getParameter("email");
       String endereco = req.getParameter("endereco");
       // instancia objeto Aluno
       Aluno aluno = new Aluno();
       // ...
       dao.insere(aluno);
       req.setAttribute("msg", "Aluno: " + aluno.getNome()
                   + " inserido com sucesso!");
       RequestDispatcher rd = req.getRequestDispatcher("/sucesso.jsp");
       rd.forward(req, resp);
   }
}
```

Separando a lógica de negócio do Servlet

- O ServletController:

```
@WebServlet("/controller")
public class ControllerServlet extends HttpServlet {

   @Override
   protected void doPost(HttpServletRequest req, HttpServletResponse
       resp) throws ServletException, IOException {

       String opcao = req.getParameter("opcao");
       String nomeDaClasse = "br.edu.ufabc.progradweb.acao." + opcao;

       try {
          Class<?> classe = Class.forName(nomeDaClasse);
          Acao acao = (Acao) classe.newInstance();
          acao.executa(req, resp);
       } catch (Exception e) {
          throw new ServletException("Erro na lógica de negócios",e);
       }
    }
 }
```

Projeto Prático – Etapa 1

Projeto Prático

- **Objetivo:**
  - Desenvolvimento de uma aplicação web dinâmica utilizando as tecnologias / conceitos vistos nas aulas

- Será desenvolvido em duas etapas:
  - **Etapa 1:**
    - Definição do tema, objetivos e funcionalidades da aplicação
    - Planejamento inicial do banco de dados e criação de algumas tabelas
    - Desenvolvimento de algumas páginas estáticas (página inicial, informações do site, etc.)
    - Desenvolvimento de algumas páginas dinâmicas (com acesso ao banco de dados) e utilização de servlets e JSP
  - **Etapa 2:** projeto final completo, empregando outros conceitos que serão vistos na outra metade do curso (filtros, Spring MVC, Hibernate, etc.)

Projeto Prático

- **Entregar na Etapa1:**
  - Relatório com descrição do tema escolhido, objetivos e funcionalidades da aplicação web; uma descrição das funcionalidades implementadas nesta etapa
  - Planejamento inicial do banco de dados (esquema)
  - Código fonte
  - Colocar tudo em uma pasta, compactar e submeter no Tidia
  - Prazo de entrega: 12/07 (domingo)

- **Apresentação: 14/07 (3a feira)**
  - 10~15 minutos aproximadamente

Avaliação do projeto

- Na avaliação do projeto serão considerados diversos aspectos como:
  - Prazos de entrega
  - Funcionalidade do projeto, complexidade
  - Implementação da aplicação, organização do código (formatação, comentários, etc.)
  - Utilização dos conceitos vistos (servlets, JSP, taglibs, MVC)
  - Texto, ortografia, formatação e coerência do relatório
  - Apresentação do projeto
  - Criatividade, inovação, tema escolhido

<u>Atenção</u>: Códigos obtidos externamente, como por exemplo, da Internet, da empresa onde trabalha, etc. serão considerados como plágio e estão proibidos.

Alguns exemplos de temas

- Biblioteca virtual
- Educação (e-learning)
- e-commerce
- Controle de estoque
- Agência de turismo
- Site de anúncios (classificados)
- Site de pessoas ou animais perdidos
- Site para doação de alimentos, remédios, roupas para situações de emergência
- Site de reclamações de problemas na cidade, como: falta de iluminação na rua, vazamento de água, buracos nas ruas, queda de árvores, etc.

Referências

- Ralph F. Grove. Web-based Application Development. Ed. Jones and Bartlett, 2010.
- Bryan Basham, Kathy Sierra, Bert Bates. Servlets e JSP. Alta Books Editora, 2010.
- Tutoriais do Java EE (site da Oracle):

  http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html