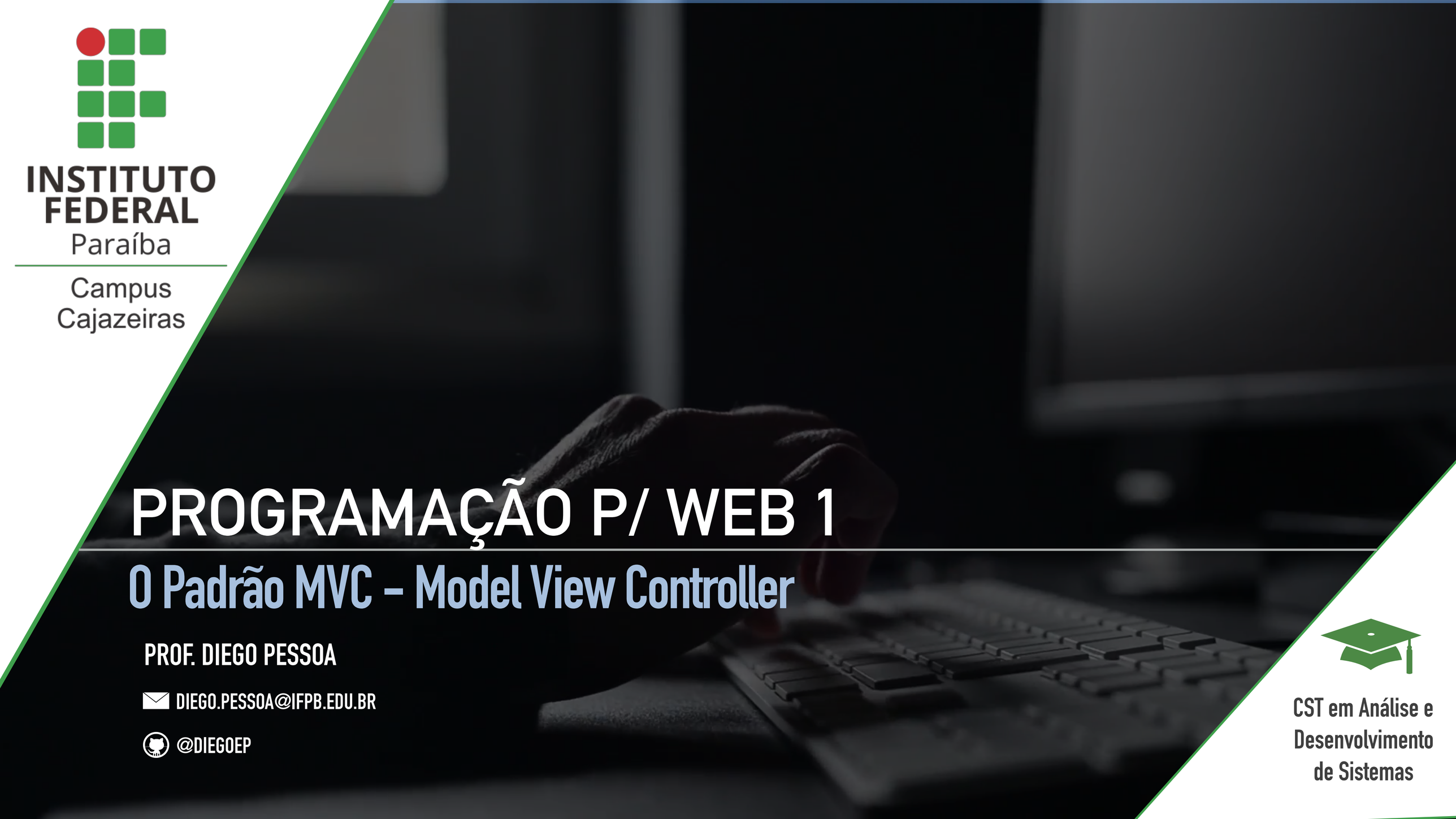


INSTITUTO FEDERAL
Paraíba
Campus Cajazeiras

PROGRAMAÇÃO P/ WEB 1

O Padrão MVC - Model View Controller

PROF. DIEGO PESSOA

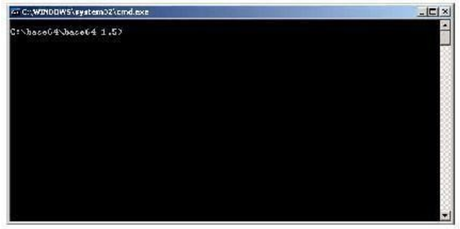
DIEGO.PESSOA@IFPB.EDU.BR

@DIEGOEP

CST em Análise e Desenvolvimento de Sistemas

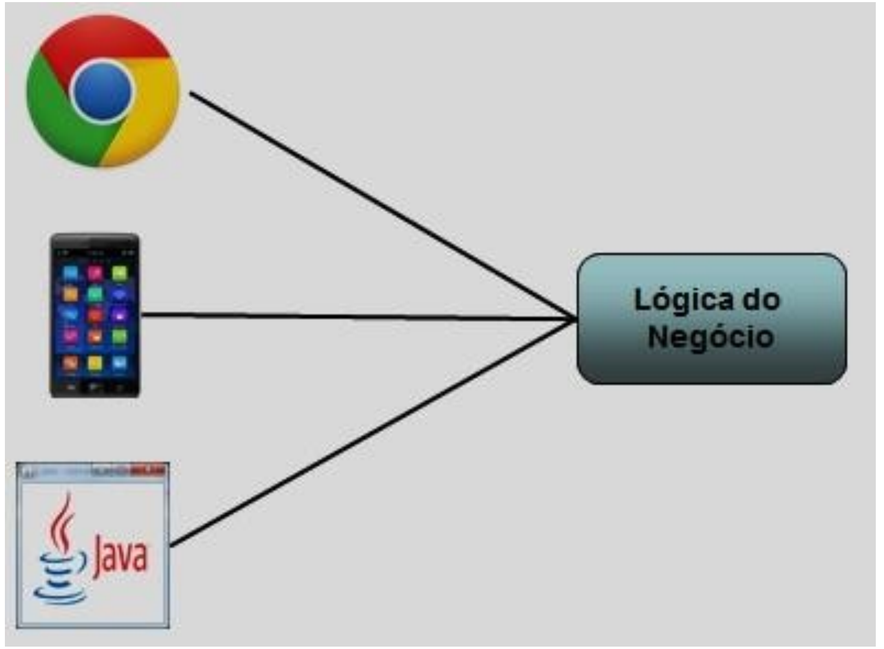
Introdução

- Aplicações Web são atualmente a principal forma de oferecer uma interface para a criação de aplicações (incluindo ambientes de Intranets)
- No entanto, é cada vez mais comum que uma aplicação necessite oferecer suporte a vários tipos de interface gráfica
  - Dispositivos móveis, aplicações Desktop, linha de comando etc.



Introdução

- Uma aplicação web bem projetada deve permitir que novos tipos de interface sejam adicionados com facilidade;
- Para isso, é importante que a lógica do negócio da aplicação seja desacoplada das interfaces de visualização;



Princípios de Projeto da Aplicação Web

- **1) Uma aplicação Web robusta separa a lógica do controle e a chamada dos objetos da camada de negócio da apresentação.**
  - Estas duas tarefas são executadas por um controlador
  - A apresentação é implementada por componentes de visão, como as páginas JSP, html, etc.

Princípios de Projeto da Aplicação Web

- **2) A aplicação Web deve ser o mais leve possível**
  - Não deve conter mais código Java do que o necessário para iniciar os processo de lógica de negócio e mostrar os resultados obtidos;
  - Não deve conhecer detalhes sobre a lógica de negócio da aplicação

Exemplo Motivacional

- Para ilustrar a aplicação do padrão MVC em aplicações web, vamos usar uma aplicação bastante simples;
- A aplicação é uma loja de livros online, que oferece três tipos de busca: por título, por autor e por ISBN;

Problemas de Projeto (Design Issues)

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException{
    String title = request.getParameter("query");
    BookStoreDaoFactoryIF bs = BookStoreDaoFactory.createBookStoreDaoFactory();
    BookDaoIF bookDao = bs.createBookDao();
    try {
        List<Book> books =  bookDao.searchByTitle(title);
        Iterator<Book> it = books.iterator();
        PrintWriter out = response.getWriter();
        out.println("<html> <body> <p> Resultado da busca por "+title+"</p>");
        while(it.hasNext()){
            Book currentBook = it.next();
            out.println("<p> Título: "+currentBook.getTitle() +"</p>");
            out.println("<p> Autores: "+currentBook.getAuthors() +"</p>");
            out.println("<p> Editora: "+currentBook.getPublisher() +"</p>");
            out.println("<p> Ano:"+currentBook.getPublicationYear() +"</p>");
            out.println("<p> ISBN: "+currentBook.getIsbn() +"</p>");
            out.println("<p> Preço: "+currentBook.getPrice() +"</p>");
        }
        out.println("</body></html>");
    }
    catch (PersistenceException ex) {
    }
```

Exemplo Motivacional

- **Problemas:**
  - Trecho 1: O servlet conhece detalhes sobre a implementação da lógica de negócio da aplicação;
  - Trecho 2: O servlet também é responsável por gerar a página que vai apresentar os resultados;

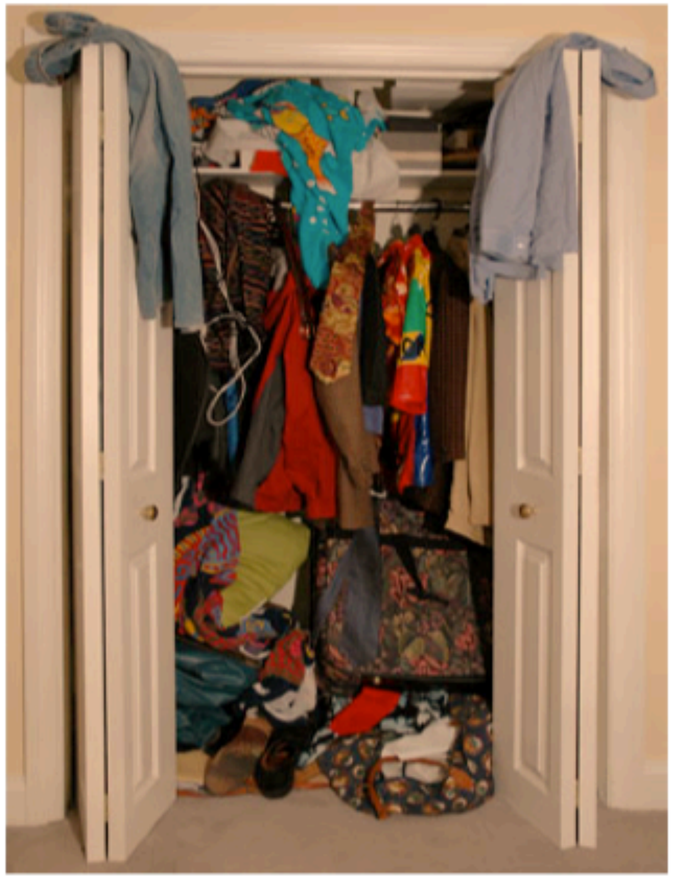
Exemplo Motivacional

- Agora, imagine o que aconteceria:
  - Se quiséssemos oferecer aos nossos clientes uma outra interface, como linha de comando, desktop, etc;
  - Se quiséssemos mudar o layout da página que apresenta os resultados;
  - Se precisássemos alterar a implementação da lógica de negócio;

O Padrão MVC

- A solução para estes problemas é o padrão MVC;
- O padrão MVC isola a interação entre os componentes de visão e os objetos das regras de negócio da aplicação;
- A interação é dividida em três camadas:
  - Visão;
  - Modelo;
  - Controlador;

EXEMPLO DA VIDA REAL...



Guarda-roupa em camada única

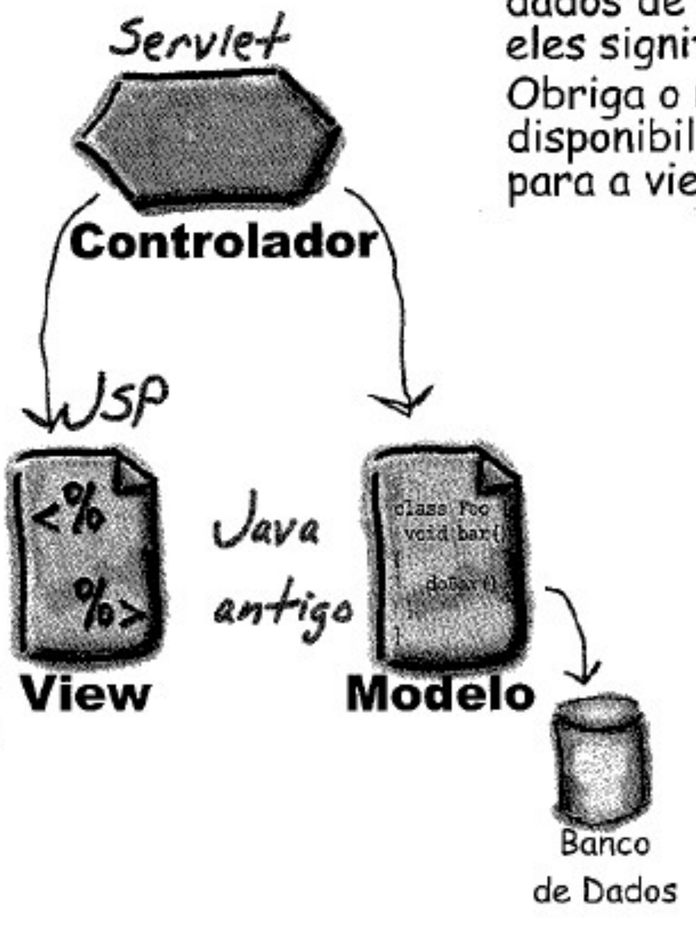
Guarda-roupa em camadas distintas

O Padrão MVC

- O padrão MVC permite que a lógica de negócio da aplicação seja acessada a partir de vários tipos de interface;
- Permite que novos tipos de interface sejam adicionados com mais facilidade;
  - E facilita a manutenção dos objetos da lógica de negócio;

O MVC no mundo Servlet & JSP

CONTROLADOR
Retira da solicitação do usuário os dados de entrada e interpreta o que eles significam para o modelo.
Obriga o modelo a se atualizar e disponibiliza o estado do novo modelo para a view (o JSP).



VIEW
Responsável pela apresentação. Ela recebe o estado do modelo do Controlador (embora não diretamente; o Controlador põe os dados do modelo em um lugar onde a View possa encontrá-lo). Também é a parte que recebe os dados de entrada do usuário que volta ao Controlador.

MODELO
Abriga a verdadeira lógica e o estado do modelo. Em outras palavras, ele conhece as regras para obtenção e atualização do estado.
O conteúdo de um Carrinho de Compras (e as regras sobre o que fazer com isso) seria parte do Modelo no MVC.
É a única parte do sistema que se comunica com o banco de dados (embora ele provavelmente use outro objeto para a verdadeira comunicação com o DB, mas guardaremos este padrão para mais tarde...).

MVC – as Funções do Modelo (M)

- Os objetos do modelo implementam as regras de negócio da aplicação;
- Os seus objetos fornecem os dados que serão trocados entre as camadas e usados na camada de visão;

MVC – as Funções da Visão (V)

- A camada de visão é responsável por coletaros dados passados pelo cliente;
- As ações que devem ser executadas com seus respectivos parâmetros de entrada;
- Ela também é responsável por apresentar o estado atual do modelo para o usuário;

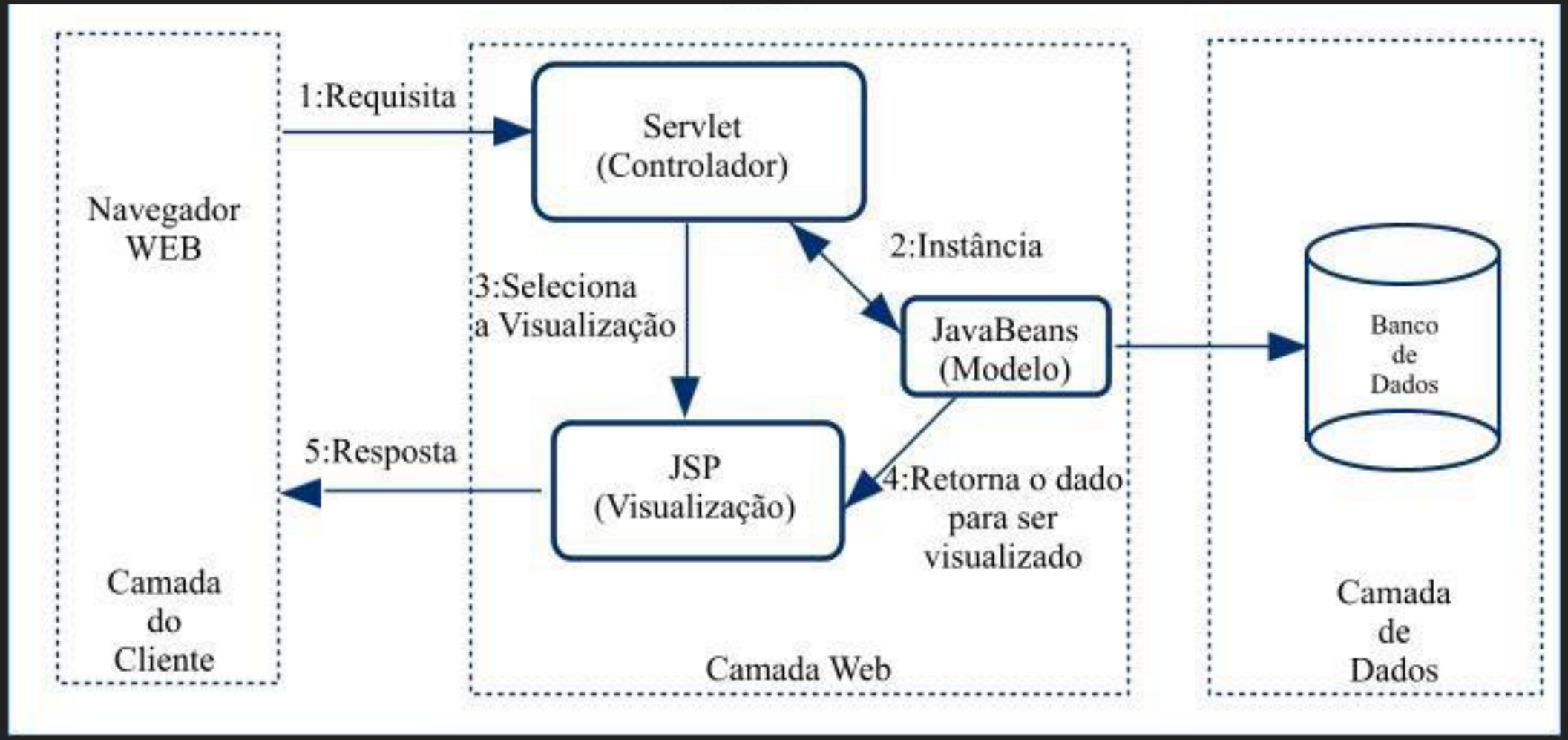
MVC – as Funções do Controlador (C)

- O controlador é responsável por processar os valores de entrada passados pelo cliente;
  - E transformar a requisição do cliente em ações nos objetos do modelo;
- O controlador pode criar um novo objeto do modelo ou usar um objeto já instanciado;

MVC com Servlets e JSP

- Implementação do padrão MVC em uma aplicação Web Java:
  - O modelo é implementado através de classes java tradicionais (POJO)
  - A lógica de controle é implementada através de servlets;
  - A visão é implementada usando JSP.

MVC NA CAMADA WEB EM JAVA



Implementando o Modelo

- O principal objetivo do padrão MVC consiste em fazer com que a lógica do negócio da aplicação possa ser reusada e acessada por diferentes clientes;
- Para isso, isolamos toda a lógica do negócio em classes java;
- Os padrões de projeto Facade e AbstractFactory podem ser usados para desacoplar ainda mais o modelo da camada de controle;
- O controlador só conhece a interface ou a classe principal da aplicação com a qual ele vai interagir;

EXEMPLO DE IMPLEMENTAÇAO DO MODELO

```
public class BookStoreApplication  {

    private BookDaoIF bookDao;
    private static BookStoreApplication instance = null;

    private BookStoreApplication(){...4 linhas }

    public static BookStoreApplication getInstance(){...6 linhas }

    public List<Book> searchByTitle(String title)
            throws BookApplicationException{...8 linhas }

    public Book searchByISBN(String isbn)
            throws BookApplicationException{...8 linhas }

    public List<Book> searchByAuthor(String author)
            throws BookApplicationException{...8 linhas }


}
```

Implementando o Modelo

- Note que os controladores terão que interagir apenas com uma classe para executar as regras de negócio;
- Mudanças realizadas em outras classe da lógica de negócio não afetam os controladores da aplicação;
- A manutenção da camada fica bem mais simples;

Implementando os Controladores

- O comportamento genérico de um servlet controlador é dividido em quatro etapas:
  - 1. Recuperar os valores dos parâmetros da requisição;
  - 2. Executar o modelo;
  - 3. Colocar os objetos do modelo na requisição;
  - 4. Encaminhar a requisição para a página JSP responsável pela visualização;
- A aplicação pode usar vários controladores para tratar os diferentes tipos de requisição que podem ocorrer na aplicação;

EXEMPLO DE IMPLEMENTAÇÃO DE UM SERVLET CONTROLADOR

```
@WebServlet("/searchByTitle")
public class SearchByTitleServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request,HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException{
        //Manipulando os parâmetros da requisição
        String title = request.getParameter("query");
        RequestDispatcher dispatcher = null;
        if(title==null){
            dispatcher = getServletContext().getRequestDispatcher("emptyQueryError.jsp");
        }
        //Interagindo com o modelo
        BookStoreApplication bookStore = BookStoreApplication.getInstance();
        try {
            List<Book> books =  bookStore.searchByTitle(title);
            //Invocando o componente de visão
            request.setAttribute("books", books);
            dispatcher = getServletContext().getRequestDispatcher("showBooks.jsp");
            dispatcher.forward(request, response);
        }
        catch (BookApplicationException ex) { response.sendError(555);}
    }
}
```

Implementando o Controlador

- Note que o servlet recupera a instância do modelo e executa a operação solicitada, mas não conhece os detalhes da sua implementação;
  - Mudanças na implementação da lógica de negócio do modelo não requerem alteração no servlet controlador;
- Note que o servlet sabe qual componente da visão será responsável por apresentar os resultados, mas também não conhece nada sobre o layout da apresentação;
  - Mudanças no layout da página não requerem qualquer alteração no servlet controlador;

Implementando a Visão

- Finalmente, a camada de visão é responsável apenas por apresentar o resultado ao usuário, usando JSP;
- A página obtém os dados que devem ser apresentados a partir dos atributos embutidos na requisição;
- A visão representa o front-end da aplicação;

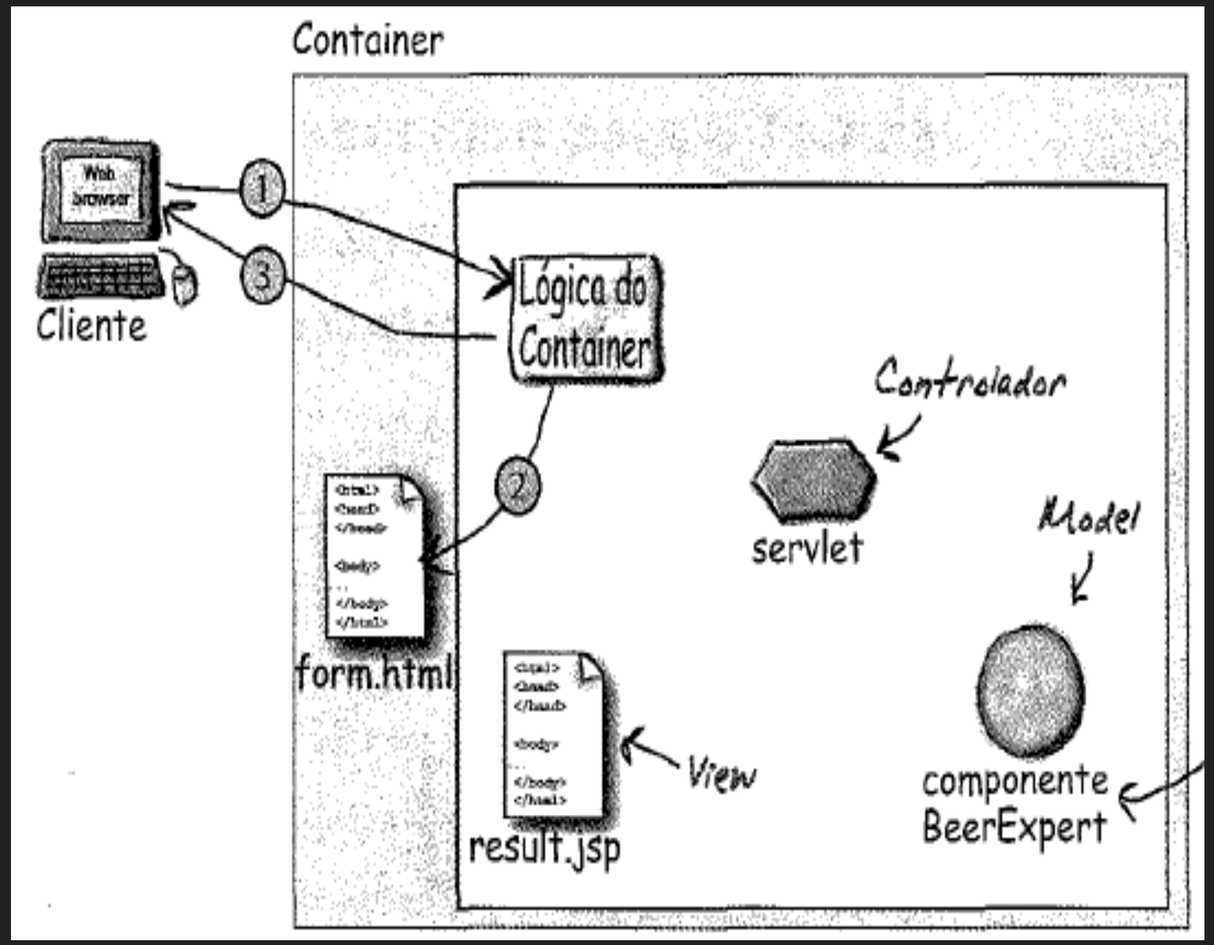
EXEMPLO DE IMPLEMENTAÇAO DA VISAO EM JSP

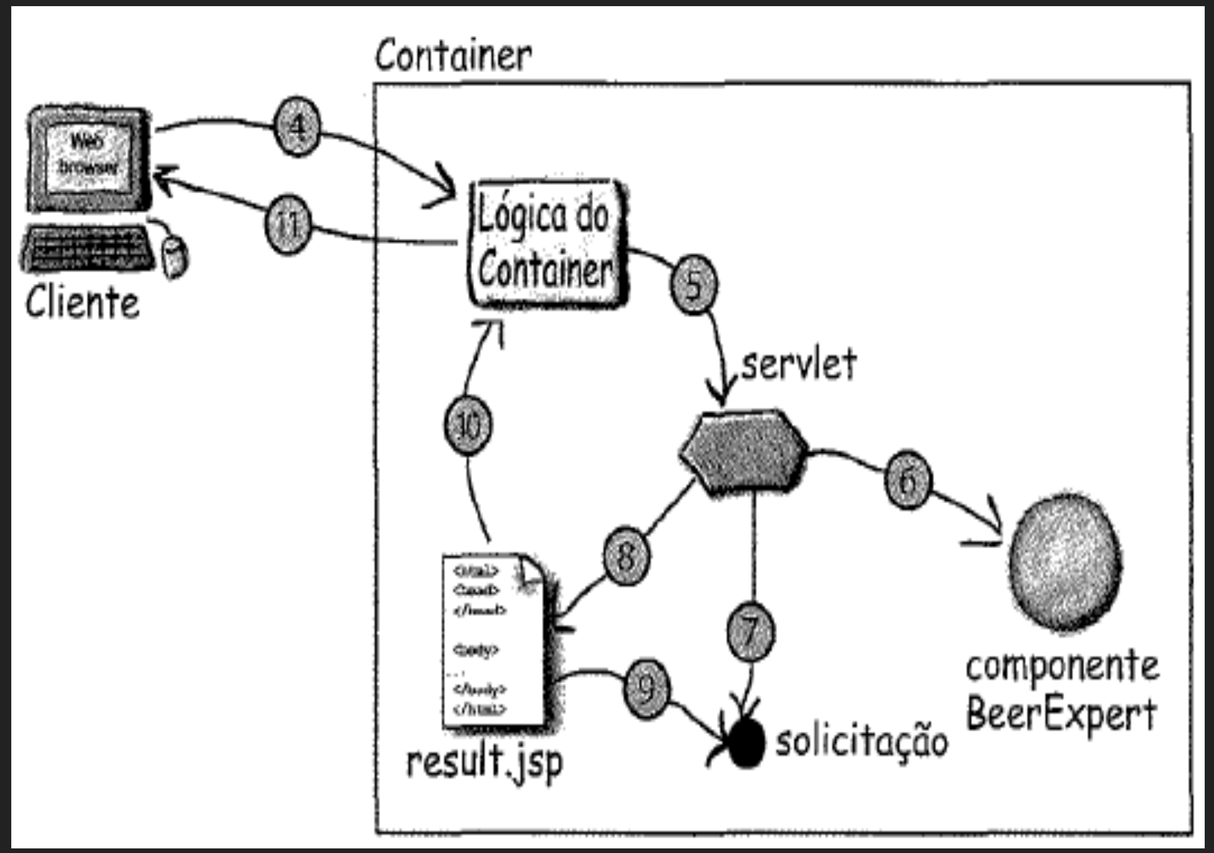
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>Book Store - Resultado da Consulta</title>
</head>
<body>
  <p> Resultado da consultas</p>
  <c:forEach var="book" items="${books}">
        <p> Título: ${book.title} </p>
        <p> Autores: ${book.authors} </p>
        <p> Editora: ${book.publisher} </p>
        <p> Ano: ${book.publicationYear} </p>
        <p> ISBN: ${book.isbn} </p>
        <p> Preço: R$ ${book.price} </p>
  </c:forEach>
</body>
</html>
```

Implementando a Visão

- Note que o desenvolvedor da página JSP que vai exibir o resultado não precisa conhecer qualquer detalhe sobre a implementação do modelo;
- Ele só precisa saber o nome do atributo no qual o resultado está armazenado e as propriedades da classe Book;

APLICANDO MVC NO PROJETO DA APLICAÇAO WEB





Aplicação MVC no Projeto da Aplicação Web

- A nossa solução MVC atual melhora o projeto da camada web, mas usa um sevlet controlador para cada operação da aplicação;
- No entanto, tal situação pode levar a alguns problemas:
  - Múltiplos pontos de acesso à camada de controle, o que dificulta o controle e o gerenciamento das requisições;
  - O código de tarefas repetidas da lógica de controle é replicado em vários servlets;

Aplicação MVC no Projeto da Aplicação Web

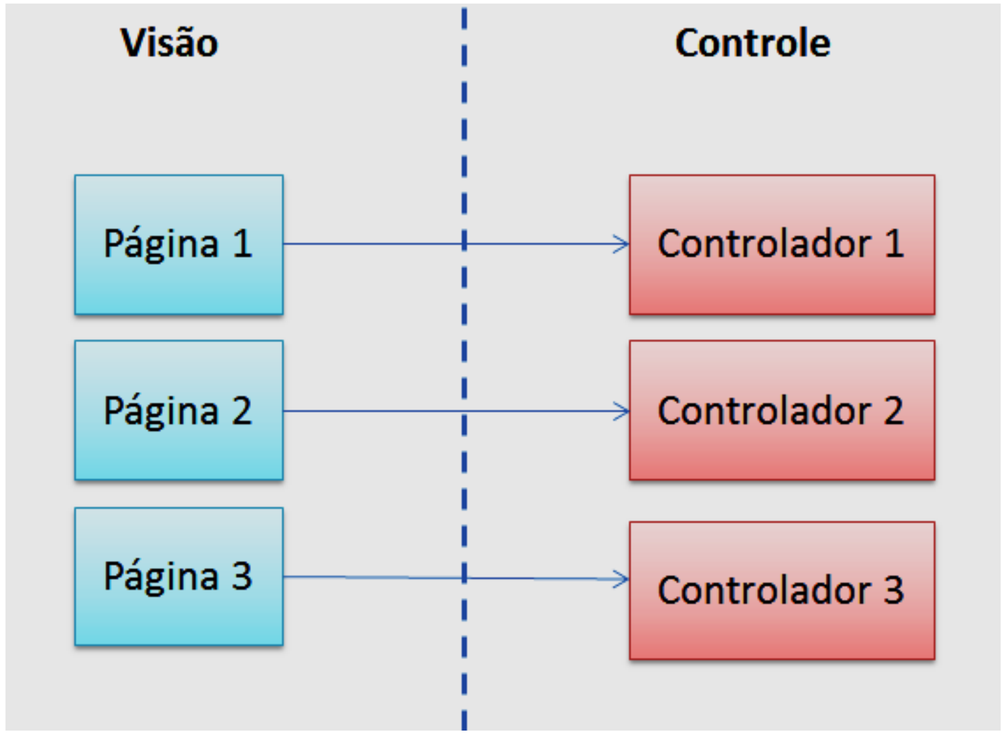
- Em algumas situações, temos que executar uma tarefa da lógica de controle em mais de um controlador;
- Por exemplo, podemos validar os dados recebidos, checar se o usuário ainda está com o login ativo, fazer logs de requisições, etc;
- Replicar esta lógica em cada controlador é ruim, pois dificulta a gerência das requisições e a manutenção da aplicação
- Imagine o que iria acontecer quando quisermos alterar esta lógica de controle.
- Criar um único servlet responsável por fazer o trabalho de todos os controladores também não é uma boa ideia; (imagine o tamanho final deste servlet)

O Padrão Front Controller

- O padrão Front Controller nos permite superar estas limitações;
- Com este padrão, implementamos um controlador único que é responsável por receber todas as requisições;
- E solicitar a invocação dos comandos solicitados pelos clientes;

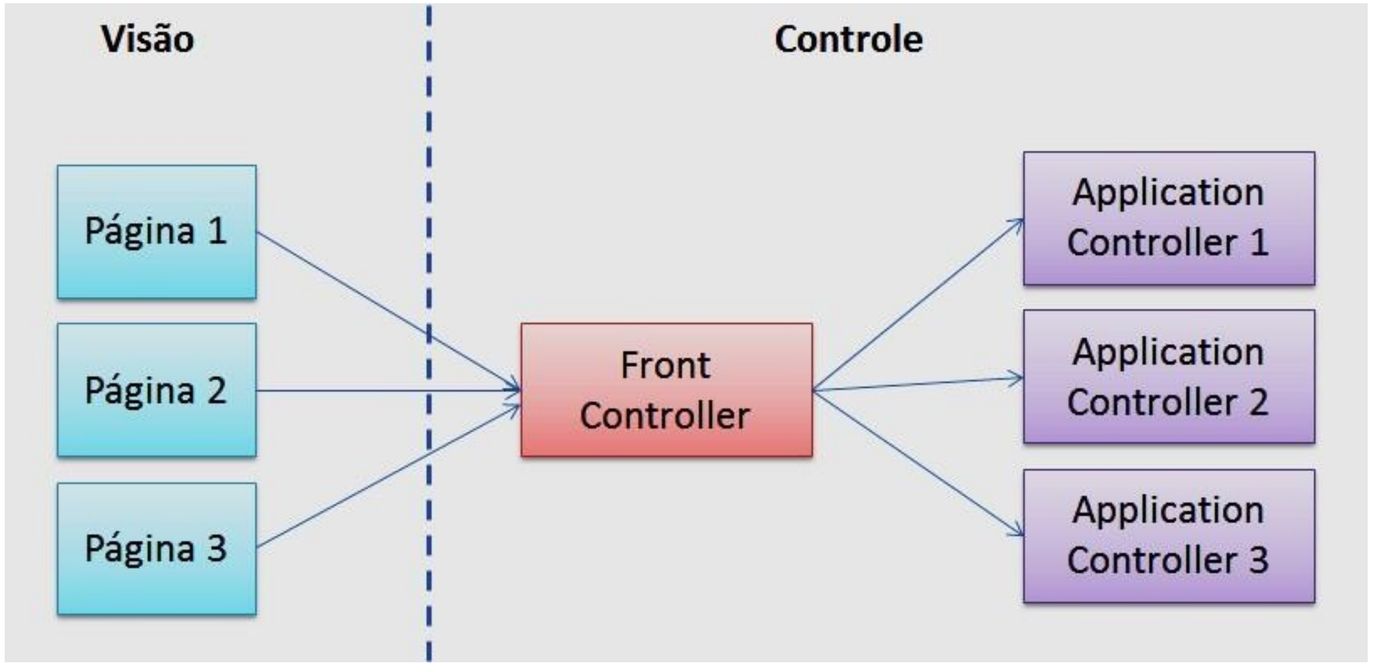
O Padrão Front Controller

- Na nossa solução atual, as requisições são processadas desta forma:



O Padrão Front Controller

- Com o padrão Front Controller, elas serão processadas da seguinte forma:

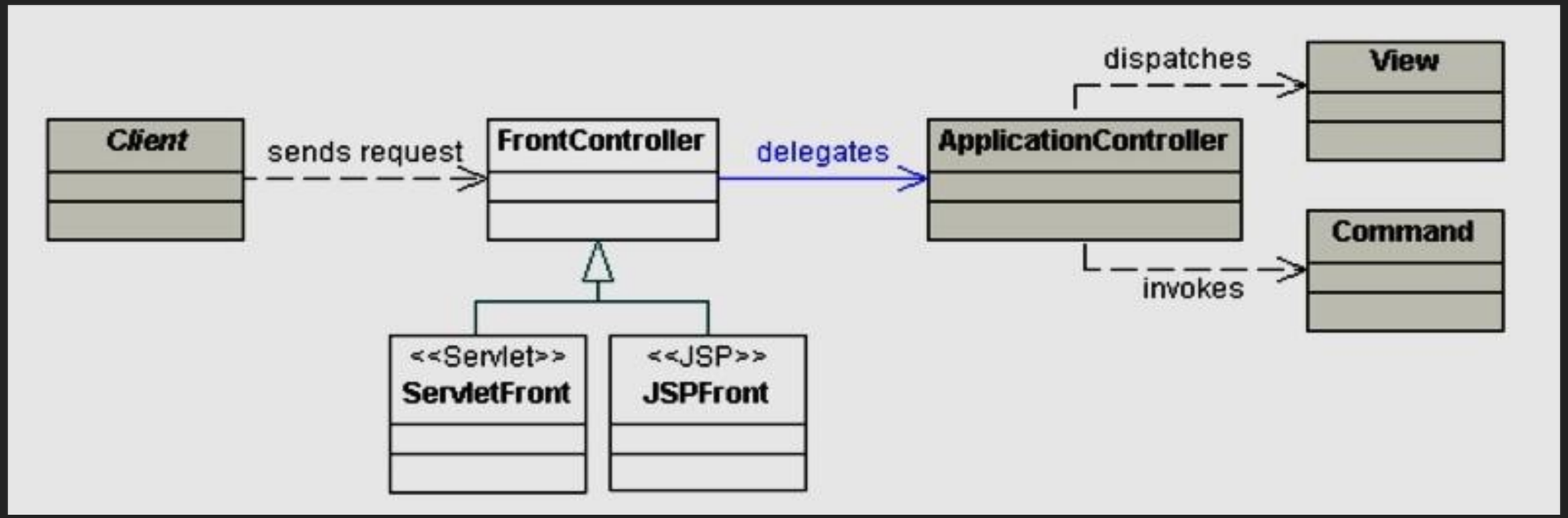


O Padrão Front Controller

- Vantagens:
  - Permite a separação da lógica de processamento da visão;
  - Centralizaos pontos de controle de acesso no sistema;
  - Evita a duplicação do código da lógica da camada de controle;
  - Permite a aplicação do código de controle a múltiplas instâncias;

O PADRAO FRONT CONTROLLER

- Diagrama de Classes:



O Padrão Front Controller

- No padrão Front Controller, o código da lógica de controle é dividido em dois componentes;
  - O Front Controller;
  - Os Application Controllers;

O Padrão Front Controller

- É o ator central do padrão e representa o componente que irá centralizar o processamento das requisições;
- É normalmente implementado através de um servlet, embora também possa ser uma página JSP;
- É responsável pelo processamento da lógica de controle que é comum a todos os controladores;
- Após executar a lógica comum, o Front Controller encaminha o controle da requisição para p Aplpication Controller adequado;

O Padrão Front Controller

- Os Application Controllers:
  - Os application controllers são responsáveis por executar o código da lógica de controle que é específico da tarefa a ser executada;
  - Este componente executa a tarefa solicitada pelo cliente no modelo;
  - E encaminha a solicitação para o componente de visão responsável por exibir os resultados;
  - Os objetos Application Controllers são normalmente implementados com o padrão **Command**.

O Padrão Command

- Padrão que nos permite encapsular o processamento das solicitações dos clientes como objetos;
  - Também é chamado de Action ou Transaction;
- Permite a especificação de uma interface comum para todas as ações executadas na aplicação;
- Reduz o acoplamento entre o Front Controller e os Application Controllers
  - Cada ação é tratada como um comando
  - O front controller só conhece a interface dos outros controladores
  - Novos tipos de controladores podem ser adicionados/removidos de forma transparente

O Padrão Command

- Para implementar o MVC Ad hoc na camada web usando o front controller precisamos:
  - Definir a interface dos comandos;
  - Implementar os comandos da aplicação;
  - Implementar o servlet do front controller;

O Padrão Command

- Definindo a interface dos comandos:
- A interface dos comandos define todos os métodos que devem ser oferecidos pelos application controllers;
- A interface deve ter pelo menos um método responsável pelo processamento da requisição;

O PADRAO COMMAND

- Definindo a interface dos comandos:

```
public interface Command {

    public void execute(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, BookApplicationException, IOException;

}
```

O Padrão Command

- Implementando os comandos da aplicação:
  - Uma aplicação web normalmente é composta por várias implementações da interface Command;
  - Cada implementação é responsável por executar uma determinada ação no modelo;

O Padrão Command

- Tarefas comuns na implementação do método de processamento de um comando:
  - Criar/recuperara instância do modelo que deve ser usada para executar a ação;
  - Invocar a ação;
  - Colocar os resultados obtidos em um local em que os mesmos possam ser recuperados pelo componente de visão;
  - Encaminhar a requisição do cliente para o componente da visão adequado;

O PADRAO COMMAND

- Implementando os comandos da aplicação:

```
public class SearchByTitle implements Command{

    public void execute(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, BookApplicationException, IOException {
        String title = request.getParameter("query");
        BookStoreApplication bookStore = BookStoreApplication.getInstance();
        List<Book> books =  bookStore.searchByTitle(title);
        request.setAttribute("books", books);
        RequestDispatcher dispatcher = request.getServletContext().getRequestDispatcher("showBooks.jsp");
        dispatcher.forward(request, response);
    }

}
```

Implementando o Front Controller

- Uma aplicação web normalmente é composta por um único servlet que atua como front controller;
- O front controller é responsável por receber as requisições dos clientes e invocar o comando adequado;
- Tarefas comuns na implementação do front controller:
  - Executar a lógica de controle que é comum aos controladores da aplicação;
  - Identificar e instanciar o comando que deve ser executado;
  - Solicitar a execução de um Application Controller;

IMPLEMENTANDO O FRONT CONTROLLER

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

    protected void doGet(HttpServletRequest request,HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        String title = request.getParameter("query");
        RequestDispatcher dispatcher = null;
        if(title==null){
            dispatcher = getServletContext().getRequestDispatcher("emptyQueryError.jsp");
        }
        try {
            String action = request.getParameter("command");
            Command command = (Command)Class.forName("commands."+action).newInstance();
            command.execute(request, response);
        }
        catch (Exception ex) {}
    }

}
```

Implementando o Front Controller

- Note que:
  - O front controller agora é o responsável pela validação do parâmetro da consulta, o que é feito em todas as requisições;
  - O front controller instancia o comando dinamicamente;
  - O front controller conhece apenas a interface do comando executado;

Considerações Finais

- As vantagens do uso do padrão MVC na camada Web:
  - A lógica do controle é implementada dentro dos servlets;
    - Isto reduz a quantidade de código java nas páginas JSP;
- É possível centralizar a lógica de controle da aplicação em uma única camada;
- O controlador é responsável por escolher a página que vai apresentar os resultados obtidos;
  - Caso alguma exceção seja lançada pelo controlador ou pelo modelo, ela pode ser capturada antes da exibição dos resultados;
- A implementação da visão só precisa se preocupar com a visualização, reduzindo a quantidade de scriplets nas páginas;
- O Padrão front controller nos permite um melhor controle das ações executadas na lógica de controle

Considerações Finais

- Alguns frameworks disponíveis implementam já implementam o MVC para aplicações web java:
  - Struts;
  - Spring MVC;
  - Java Server Faces;
  - VRaptor;
  - Tapestry;
  - Play Framework

Considerações Finais

- O Padrão MVC pode ser facilmente usado para a implementação de aplicações Web flexíveis;
- Esta possibilidade traz importantes vantagens para o desenvolvimento de aplicações.
- Entretanto, o padrão aumenta a complexidade do projeto da aplicação Web
- Cabe ao desenvolvedor avaliar se o padrão deve ou não ser aplicado durante o desenvolvimento do projeto
- Algumas variáveis: complexidade, experiência da equipe, prazos de entrega, natureza e características dos projetos e necessidade de manutenções futuras.

Desafio

- Implementar o padrão Front Controller usando a Servlet API
  - Criar interface Command com método:
    - String execute(HttpServletRequest req, HttpServletResponse resp);
  - Criar classe de comando que implementa a interface e retorna dados para uma página JSP
  - Criar um Front Controller Servlet para receber como parâmetro o nome do comando, instanciar a classe e chamar o método execute. Em seguida, encaminhe a requisição para a página de resultado.
    - Dica: use a API Java Reflection para instanciar o comando