

Java Enterprise Edition - JEE

14. Padrões de projeto



Esp. Márcio Palheta
gtalk: marcio.palheta@gmail.com



Agenda

- Padrões de projeto
- Value Object – VO
- Singleton
- Service Locator
- Pool de conexões
- Session Facade
- Front Controller

Design Patterns – Padrões de projeto



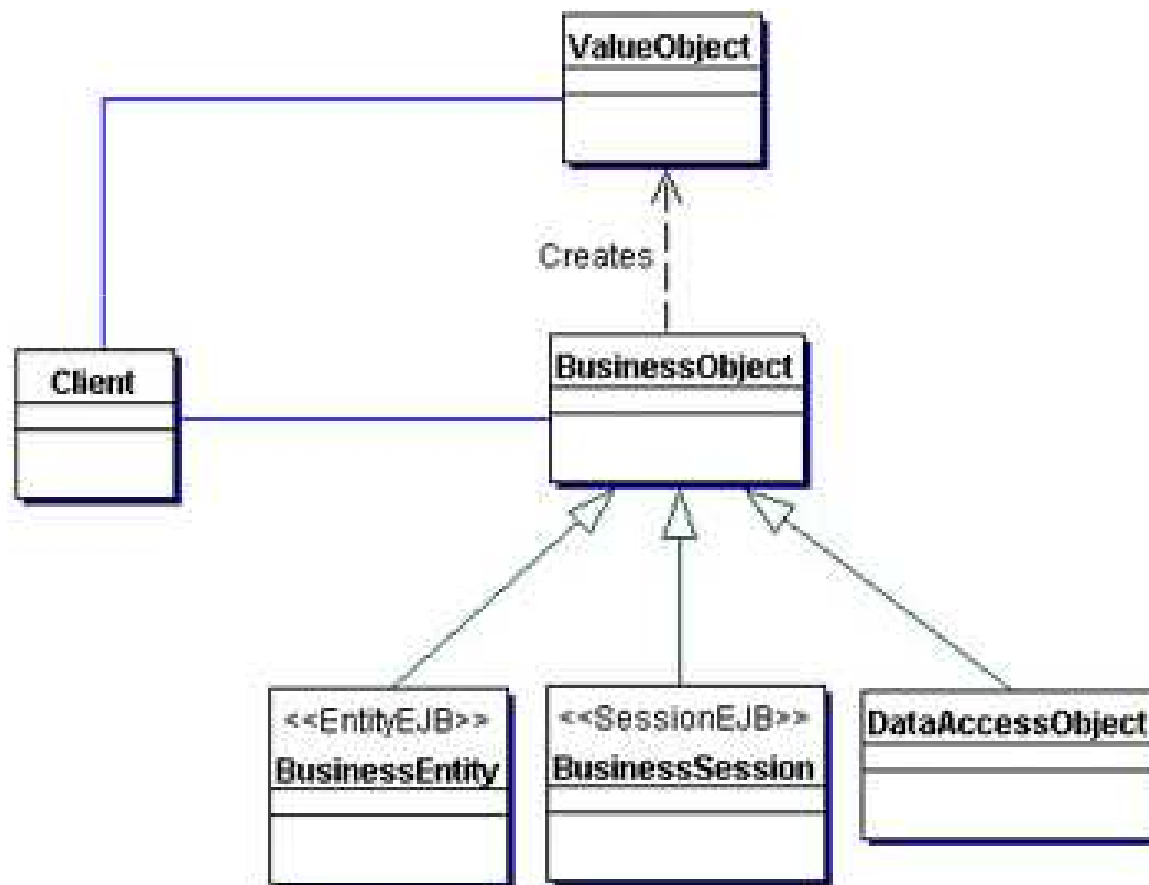
- Boas práticas oferecidas pela SUN;
- Focados em resolver um determinado problema de projetos JAVA;
- Aceitos pela comunidade POO;
- Cerca de 20 padrões oferecidos;
- <http://java.sun.com/blueprints/corej2ee/patterns/Patterns/index.html>

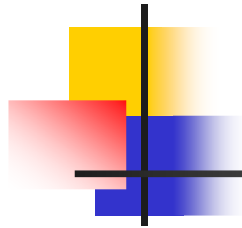


Value Object - VO

- MVC - Troca de mensagens entre camadas;
- Objetos passados por referência;
- Overhead de recursos de rede;
- **Solução:** passagem de objetos por valor;
- **Definição:**
 - Padrão de projeto utilizado para a transferência de dados entre camadas ou aplicações diferentes;

VO – Diagrama de classes





Considerações a cerca do VO

- Comumente usado quando trabalhamos com processamento distribuído;
- Implementa **Serializable** para indicar que podem ser persistidos, devendo manter seus estados;
- Conhecido, também, como:
 - Data Transfer Object – DTO; ou
 - Transfer Object – TO;



Implementação de VO

```
1 package br.fucapi.cpge.jee.model.bean;
2
3 import java.io.Serializable;
4
5 public class PessoaVO implements Serializable {
6
7     private static final long serialVersionUID = 1L;
8     private String nome;
9     private String cpf;
10    private String email;
11
12+    public String getNome() {..
15+    public void setNome(String nome) {..
18+    public String getCpf() {..
21+    public void setCpf(String cpf) {..
24+    public String getEmail() {..
27+    public void setEmail(String email) {..
30 }
```



Singleton

- Padrão de projeto que garante a existência de apenas **um objeto** de uma determinada classe;
- Centraliza o acesso a objetos especiais;
- Você não cria um novo objeto(**new**), apenas solicita uma instância já criada;
- Exemplo: Classes de fachada;

Implementação do padrão Singleton

```
public class SingletonFacade {
```

1

```
//Variável de classe
```

```
static private SingletonFacade instance = null;
```

2

```
//Retorna a instância única da classe SingletonFacade
```

```
static public SingletonFacade getInstance() {
```

```
    if(instance == null) {
```

```
        instance = new SingletonFacade();
```

```
    }
```

```
    return instance;
```

```
}
```

```
//outros métodos aqui...
```

```
}
```



Singleton - questões

- O código anterior funciona?
- Conseguimos usar o comando:
 - `new SingletonFacade()`?
- O que pode ocorrer em ambientes multithread?
- Como evitar problemas de sincronização?



Implementação Singleton

```
public class SingletonFacade {  
    1 //Variavel de classe  
    static private SingletonFacade instance;  
  
    2 //Bloco estatico  
    static{  
        instance = new SingletonFacade();  
    }  
  
    3 //Evita a criacao do construtor padrao  
    private SingletonFacade() {  
        //carga de variáveis ou configurações  
    }  
  
    4 //Retorna a instância única da classe Singleton  
    public static SingletonFacade getInstance(){  
        return instance;  
    }  
}
```



Service Locator

- Localização e criação de objetos remotos envolvem operações que consomem recursos de rede e interfaces;
- Todos os clientes necessitam desse recurso;
- Mudanças de **nome** ou **localização** exigem mudança em todas as chamadas clientes;
- Quanto mais clientes acessando, mais recursos alocados;



Service Locator - solução

- Centralizar a busca por serviços;
- Vantagens:
 - Baixo overhead na rede e no cliente final;
 - Baixo acoplamento;
 - Reuso de componentes;
 - Diminui o trabalho de manutenção;
- A seguir, mostraremos o uso de Service Locator para criação de conexões de acesso a banco de dados;



Pool de conexão

- DBCB – DataBase Connection Pool
- O problema;
- As vantagens;
- A seguir, passos para a criação do pool de conexões, usando Singleton e Service Locator;



DBCP - considerações

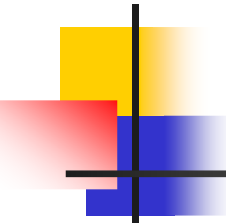
- Você deve ter uma cópia do driver do banco de dados do pool no diretório "**lib**" do Apache Tomcat.
- Você não precisa do driver na pasta WEB-INF/lib
- A seguir, são apresentados os passos necessários para implementação de pool de conexões, utilizando Service Locator



Passo para a criação do pool

- 1. No diretório META-INF da aplicação, crie um arquivo "`context.xml`";
- 2. Atualização do arquivo `web.xml`;
- 3. Criação da classe ServiceLocator, responsável pela busca de objetos DataSource, via JNDI;
- 4. Criação de uma Servlet para teste do pool de conexões;

1. WebContent/META-INF/ context.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<Context auth="Container">
  <Resource
    name="jdbc/bd_teste"
    type="javax.sql.DataSource"
    url="jdbc:mysql://localhost/test"
    driverClassName="com.mysql.jdbc.Driver"
    username="login"
    password="senha"
    maxActive="100"
    maxIdle="20"/>
</Context>
```



Variáveis

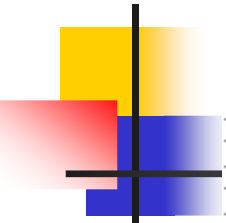
- auth → Atribui ao Apache Tomcat a responsabilidade de gerenciar a abertura e o fechamento das conexões
- name → Nome dado ao pool de conexões. Deve obedecer o formato JNDI (Mesmo nome de WEB.XML);
- type → Especifica o tipo como sendo DataSource
- url → Especifica a localização do banco de dados.
- driverClassName → Nome da classe do driver JDBC do BD
- username → Nome do usuário do BD
- password → Senha do usuário do BD
- maxActive → Número máximo de conexões ativas no pool
- maxIdle → Número máximo de conexões inativas no pool
- maxWait → Tempo máximo de espera por uma conexão, em milissegundos



2. Atualização do web.xml

```
<resource-ref>  
  <res-ref-name>  
    jdbc/bd_teste  
  </res-ref-name>  
  <res-type>  
    javax.sql.DataSource  
  </res-type>  
</resource-ref>
```

3. Classe ServiceLocator.java



```
8 public class ServiceLocator {
9     public static Connection getConexao(String JNDINome){
10         Connection connection = null;
11         InitialContext context;
12         DataSource dataSource;
13         try {
14             // Obtém a raiz da hierarquia de nomes
15             1 System.out.println("Criação do contexto...");
16             context = new InitialContext();
17             // Obtém a origem dos dados
18             2 System.out.println("Localização do datasource...");
19             dataSource = (DataSource)
20                 context.lookup("java:comp/env/" + JNDINome);
21             // Obtém uma conexão
22             3 System.out.println("Criação da conexão...");
23             connection = dataSource.getConnection();
24         } catch (NamingException e) {
25             e.printStackTrace();
26         } catch (SQLException e) {
27             e.printStackTrace();
28         }
29         System.out.println("Retorno da conexão");
30         return connection;
31     }
32 }
```

4. Servlet para teste do pool de conexões

```
14 @SuppressWarnings("serial")
15 public class TesteServiceLocator extends HttpServlet {
16     private static final String JNDINome = "jdbc/bd_teste";
17     protected void doGet(HttpServletRequest request,
18                           HttpServletResponse response)
19         throws ServletException, IOException {
20         doPost(request, response);
21     }
22
23     protected void doPost(HttpServletRequest request,
24                           HttpServletResponse response)
25         throws ServletException, IOException {
26         response.setContentType("text/html");
27         PrintWriter out = response.getWriter();
28         out.println("<h1>Teste de conexão</h1>");
29         //Realiza a tentativa de conexão
30         Connection con = ServiceLocator.getConexao(JNDINome);
31         out.println("<h1>Teste realizado com sucesso</h1>");
32     }
33 }
```



Session Facade

- Oferece uma interface de alto nível que facilita a utilização de um subsistema;
 - Ex: Home theater;
- **Não** encapsula as funcionalidades, uma vez que continuam disponíveis para acesso direto;
- Baixo acoplamento: O cliente não conhece as atualizações que ocorrem por trás da fachada;



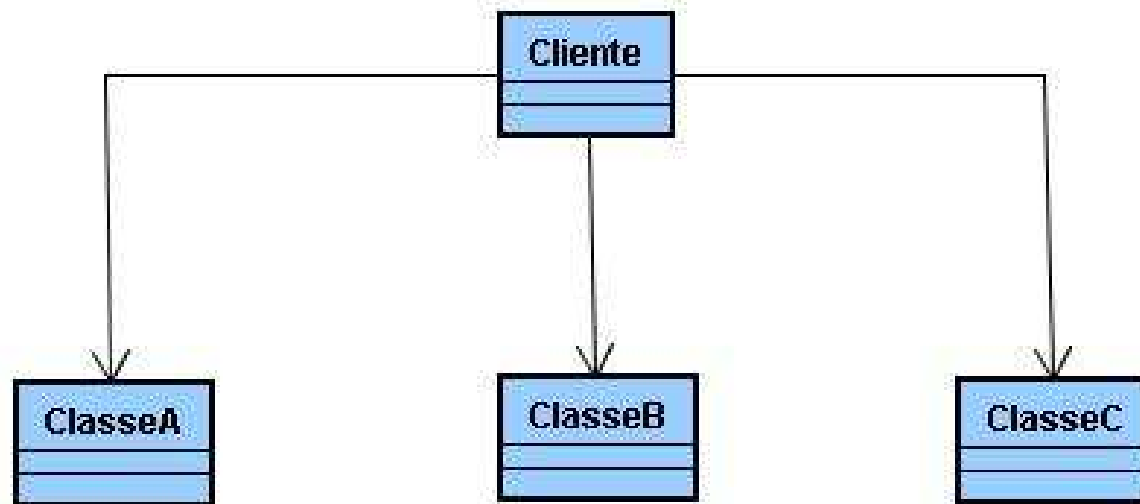
Session Facade no MVC

- Serve de interface para acesso aos serviços da camada de modelo;
- A camada de controller passa a solicitar serviços da fachada;
- A fachada fica responsável por solicitar os serviços da camada de modelo e enviar a resposta para a camada de controle



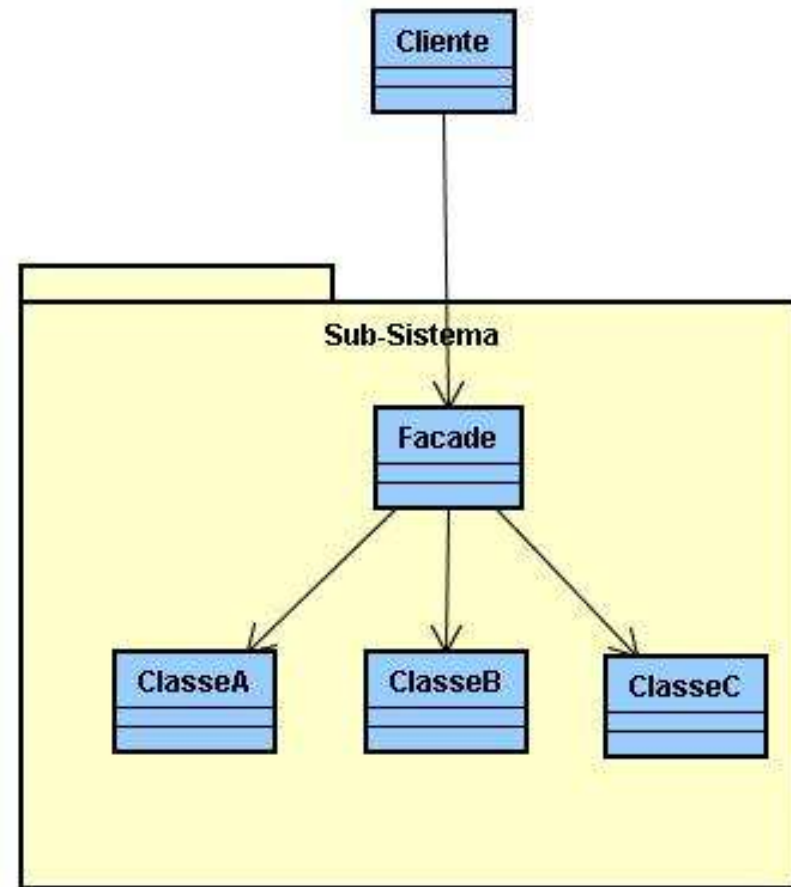
Session Facade

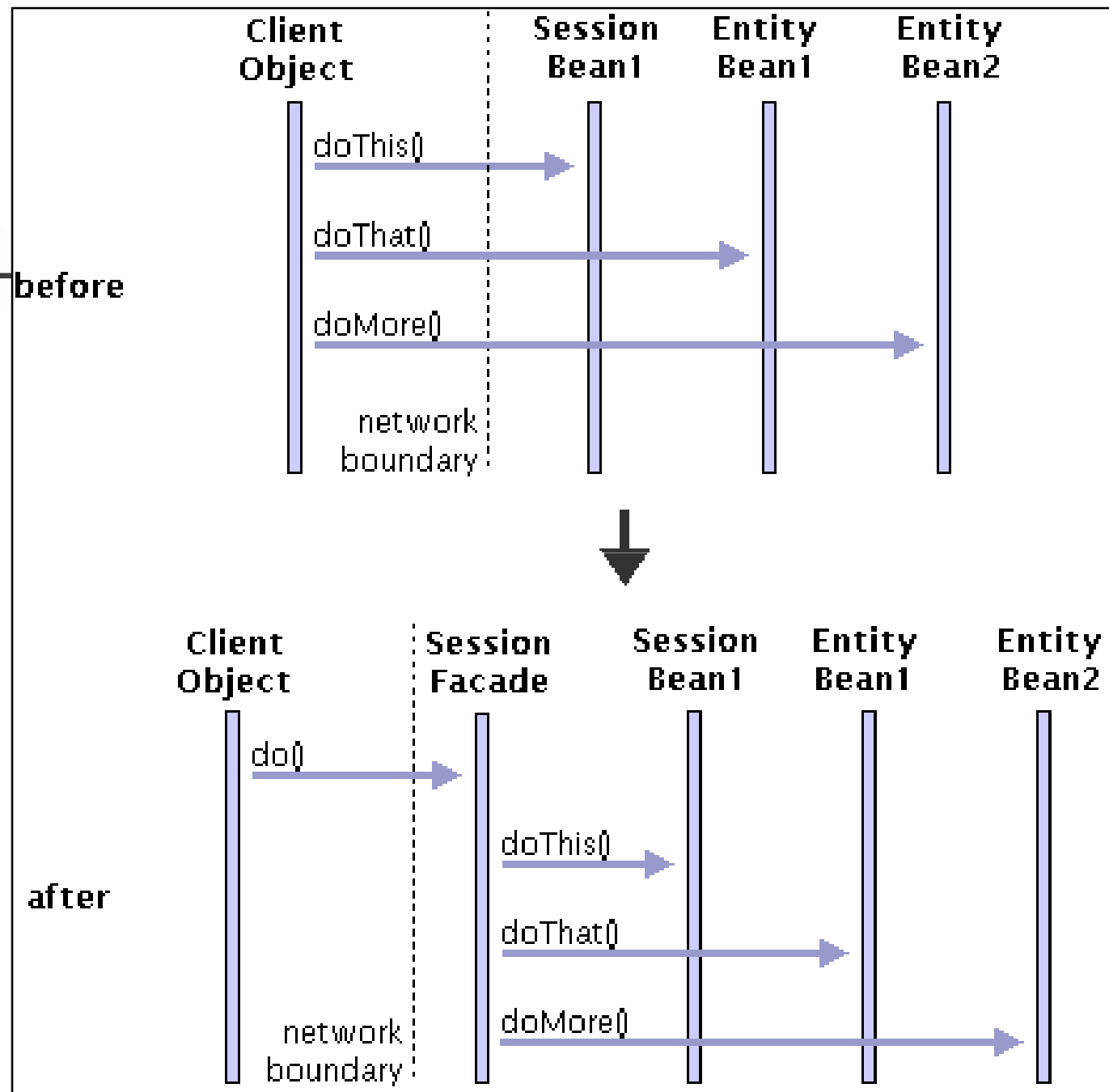
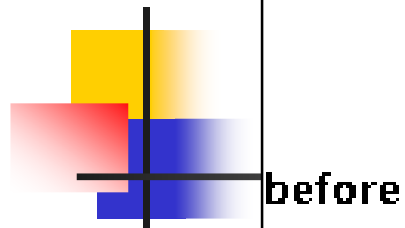
- Como tudo começou:



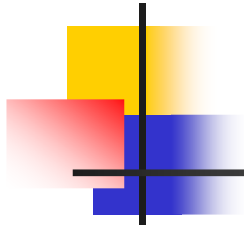
Identificação de subsistemas

- O cliente solicita Serviços da Fachada
- A fachada interage com as demais classes do subsistema





Session Facade: código



```
public class FacadeArquivo {

    private static FacadeArquivo instance;

    //***** Métodos utilitários *****
    public static FacadeArquivo getInstante(){
        if (instance == null){
            instance = new FacadeArquivo();
        }
        return instance;
    }

    //***** Métodos de Arquivo *****
    private ArquivoDAO getArquivo(){
        return ArquivoDAO.getInstance();
    }

    public Collection arquivoListar(){
        return getArquivo().listar();
    }

    public boolean arquivoCadastrar(Arquivo arquivo){
        return getArquivo().cadastrar(arquivo);
    }

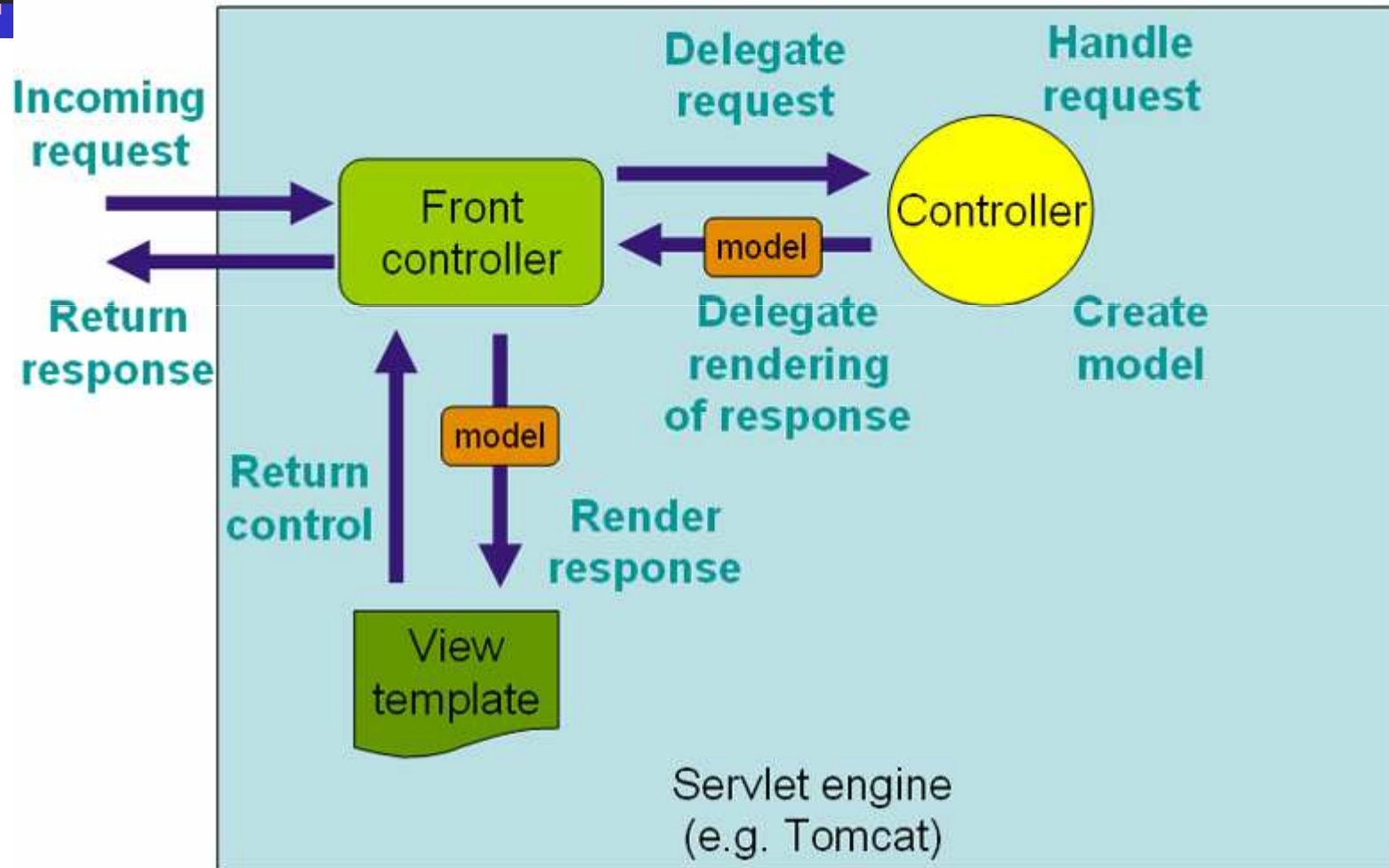
    //Outros métodos aqui
}
```



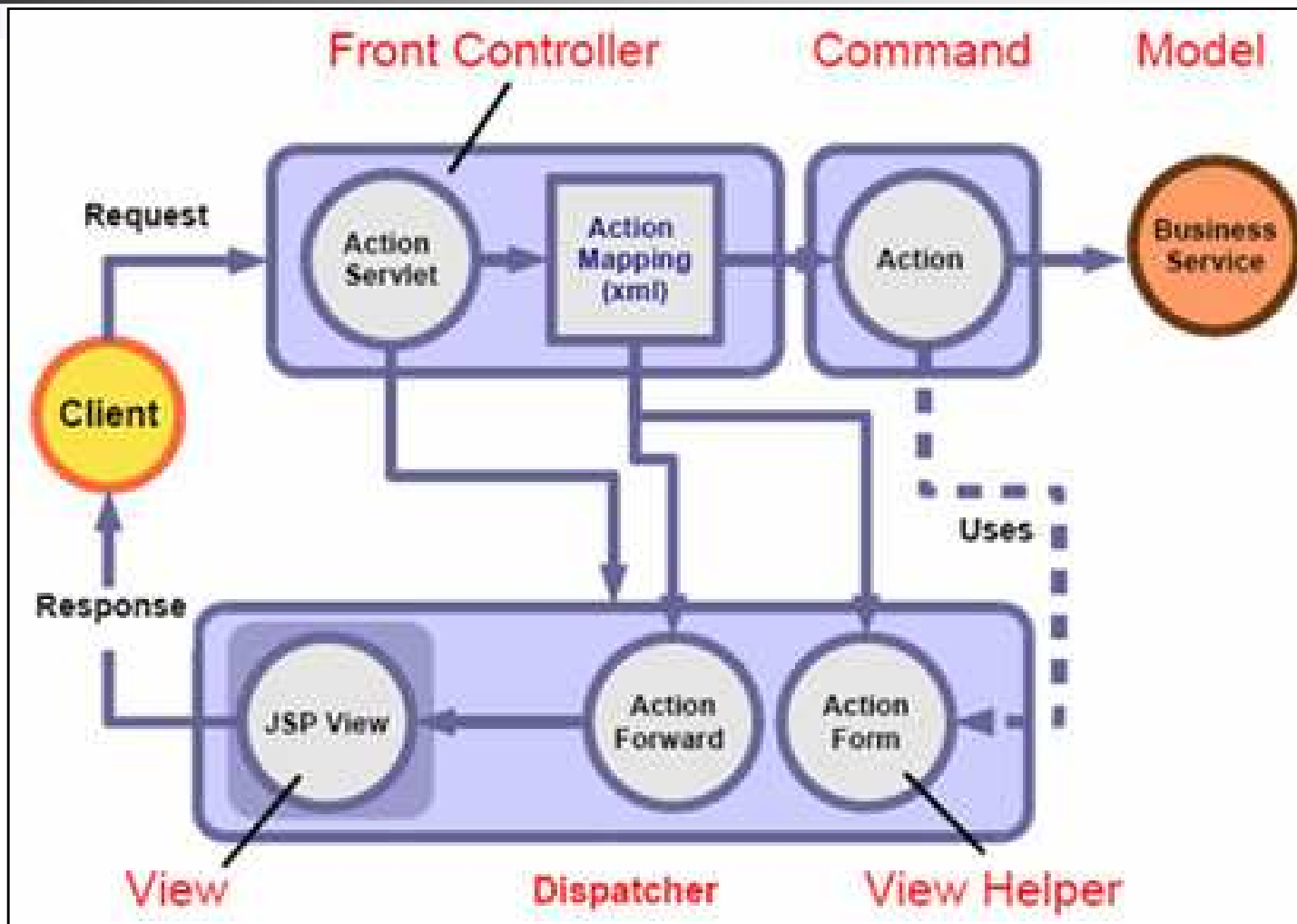
Front Controller

- Padrão de projetos que oferece um ponto centralizado para o tratamento de requisições;
- Comumente implementado em frameworks que ficam responsáveis pela camada de controle;

Front Controller



Front Controller - Struts





Atividades do projeto:

- Definição dos casos de uso;
- Definição das classes entidade;
- Definição da arquitetura;
- Criação do modelo de dados;
- Divisão do trabalho;



Referências

- Hall, Marty, “Core Servlets and Java Server Pages”, Janeiro 2002, Sun Microsystems Press;
- <http://java.sun.com/>
- <http://java.sun.com/j2ee/1.6/docs/tutorial/doc/index.html>
- <http://java.sun.com/products/jndi/docs.html>
- <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html>

Java Enterprise Edition - JEE

14. Padrões de projeto



Esp. Márcio Palheta
gtalk: marcio.palheta@gmail.com