Especializada em treinamentos Java e J2EE

- **Mais de 850 alunos treinados**
- **Mais de 4.000 em palestras e mini-cursos**
- **Instrutores certificados**
- **Em Florianópolis, na V.Office – f. (48) 224-8580**

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Palestrante

- Vinicius M. Senger
  - Trabalha com **desenvolvimento** de softwares a mais de **12 anos**;
  - Foi **instrutor** e **consultor** Java da: Sun do Brasil, Oracle e Microsoft;
  - **Palestrante** em diversos eventos nacionais e recentemente aprovado no JavaOne (maior evento Java dos Estados Unidos);
  - **Certificações**: Sun Enterprise Architect, Java Programmer 1.4, Sun Official Instructor, Oracle Instructor, Microsoft Certified Professional, Microsoft Certified Trainner;
  - Diretor Técnico e fundador da Globalcode;

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Introdução básica

- Um design-pattern é...
  - Uma forma **padrão** de organizar classes e objetos;
  - Nomes para soluções que você já modelou;
  - Uma forma de compartilhar conhecimentos sobre POO;
  - Soluções POO para problemas que incidem em diversos cenários de desenvolvimento;
  - Uma definição de conjunto finito de responsabilidades para uma classe;

Introdução básica

- Ao adotar design-patterns...
  - Seu código fica mais organizado;
  - Aumento de qualidade;
  - Menor complexidade;
  - Aumenta comunicação dentro da equipe de desenvolvimento;

Introdução básica

- A definição de um pattern pode conter...
  - **Um nome**: Transfer Object
  - **Um outro nome (also know as)**: Value Object
  - **Um problema**: algumas entidades contém dados que são sempre lidos em grupo...
  - **Uma solução**: serializar todos os dados da entidade em um objeto que...

Introdução básica

- Famílias de patterns
  - **GoF**: 23 patterns
    - *Criação*: Abstract Factory, Builder, Factory Method, Prototype, Singleton
    - *Estrutura*: Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy
    - *Comportamento*: Chain of Resp., Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor

Introdução básica

- Famílias de patterns
  - **J2EE**: Business Delegate, Composite Entity, Composite View, Data Access Object, Fast Lane Reader, Front Controller, Intercepting Filter, Model-view-controller, Service Locator, Session Façade, Transfer Object, Value List Handler, View Helper

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Patterns e Certificação

- As seguintes certificações Sun exigem conhecimentos de patterns:
  - Sun Certified Web Components Developer;
  - Sun Certified Business Component Developer;
  - Sun Certified Enterprise Architect;
- O que e quanto estudar?
  - Todos patterns J2EE;
  - Aplicar na prática os principais GoF e os mais obscuros conhecer a teoria básica;

Patterns e Certificação

- Qual das opções não é um benefício da utilização dos design-patterns:

  a) Eles fornecem uma linguagem comum para discussões sobre o design.
  b) Eles fornecem soluções para os problemas “do mundo real”.
  c) Ele comunicam a experiência obtida previamente.
  d) Eles fornecem soluções aos problemas totalmente inusitados.

Patterns e Certificação

- Qual das opções não é um benefício da utilização dos design-patterns:

  a) Eles fornecem uma linguagem comum para discussões sobre o design.
  b) Eles fornecem soluções para os problemas “do mundo real”.
  c) Ele comunicam a experiência obtida previamente.
  d) Eles fornecem soluções aos problemas totalmente inusitados.

Patterns e Certificação

- O design pattern Decorator aparece frequentemente em qual pacote Java:

  a) java.io
  b) java.awt
  c) java.lang
  d) java.util

Patterns e Certificação

- O design pattern Decorator aparece frequentemente em qual pacote Java:

a) java.io
b) java.awt
c) java.lang
d) java.util

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Singleton

- **Definição**: garantir que uma classe tenha somente uma instance.
- **Warning**: Devemos tomar cuidado com NullPointerException em servidores em cluster com Singleton implementados com static

```
public class FormatHelper {
  private static FormatHelper instance = new FormatHelper();
  ...
  public static FormatHelper getInstance() {
    return instance;
  }
  protected FormatHelper() {
  }
  public String fullDateFormat(java.util.Date data) {
    if(data==null || data.equals("")) return "";
    else return dataCompleta.format(data);
  }
}
```

Abstract Factory

- **Definição**: prover uma interface para criação de classes de uma família sem especificar a casse concreta.

- O melhor exemplo de implementação é a Home Interface do EJB

```
package br.com.globalcode.aa.ejb.session.dao;
import javax.ejb.*;
import java.rmi.*;
public interface CursosDAOHome extends javax.ejb.EJBHome {
  public CursosDAO create() throws CreateException, RemoteException;
}
```

- O cliente fica dependente apenas da interface de criação e não tem contato com o “Concrete Factory”, quem o cria é o J2EE container em deployment time.

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- **GoF patterns: comportamento**
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Command / Action

- **Definição**: encapsula uma requisição ao software em um objeto.
- Action do Struts é o principal exemplo de implementação deste pattern.

Chain of Responsability

- **Definição**: prevê uma maneira de criar um conjunto de classes que serão acionadas quando um request for enviado para um objeto.

- J2EE Servlet Filter implementa este pattern, com Filters podemos associar um conjunto de métodos “doFilter()” que serão acionados quando um request for enviado para o servidor Web.

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- **GoF patterns: estrutura**
- J2EE patterns
- Conclusões

Composite

- **Definição**: define um estrutura de objetos em formato de árvore de dados

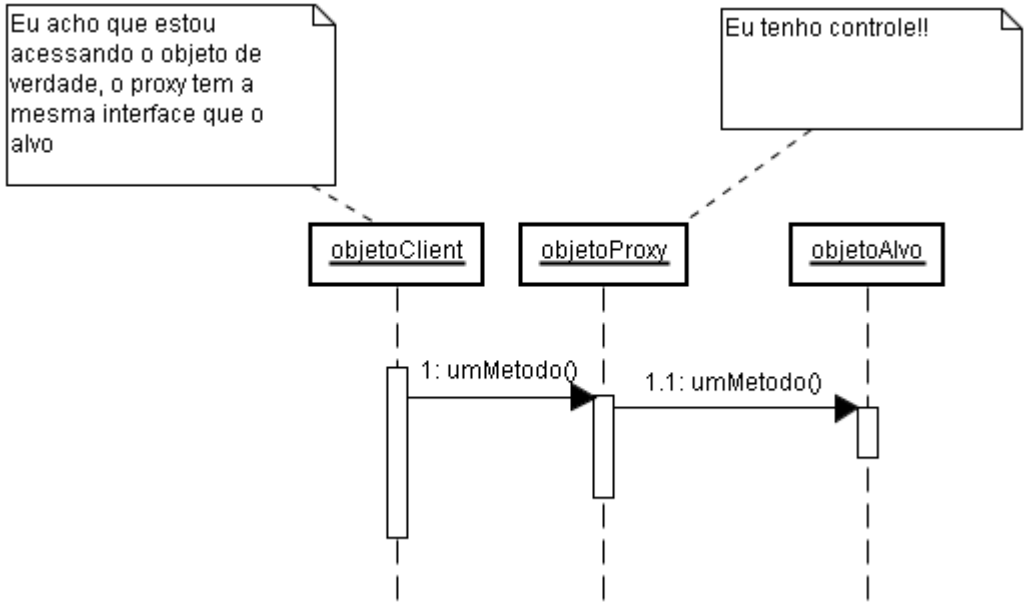
```
public class Produto {
  private Componente componente;
  public void setComponente(Componente componente) {
    this.componente = componente;
  }
  public Componente getComponente() {
    return componente;
  }
}
class Componente {
  private Componente componente;
  public void setComponente(Componente componente) {
    this.componente = componente;
  }
  public Componente getComponente() {
    return componente;
  }
}
class Motor extends Componente {}
class Cilindro extends Componente {}
```

Composite

```
class Usa {
  public void test() {
    Produto p = new Produto();
    Componente motor = new Motor();
    motor.setComponente(new Cilindro());
    p.setComponente(motor);
  }
}
```

Proxy

- **Definição**: prover um objeto intermediário para acessar outro objeto.
- O maior exemplo de Proxy em Java são Stubs e Skeletons RMI.



Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Service Locator

- **Definição**: simplifica o acesso a recursos J2EE em um aplicativo centralizando lookups JNDI em classes específicas de localização de serviços.
- Evita que sua solução tenha alto acoplamento com JNDI Naming Service;
- Tomar cuidado com Service Locator e cluster;
- Utilize sempre que possível ENC;
- Exemplo no JAREF

Data Access Object

- **Definição**: centraliza o serviço de persistência de objetos em um pequeno conjunto de classes, evitando por exemplo que código SQL se espalhe pelo código da solução.

- Mesmo utilizando framework de persistência, utilize Data Access Object

- Exemplo no JAREF

Front Controller

- **Definição**: centraliza requests em um ponto central na solução.

- No lugar de um JSP submit para outro JSP, todos os JSP's "submits" para um Servlet Front Controller que será responsável por processar as requisições.

- Exemplo no JAREF - FullController

Composite View

- **Definição**: separa uma visualização (JSP / Swing) em pequenas partes para poder reaproveitar elementos comuns a várias views.

- Include de JSP';
- Componentização de "pedaços" de telas Swing;
- Framework Tiles;

- Exemplo no JAREF

Model-view-controller

- **Definição**: divide o aplicativo em dados, comportamento e apresentação.
- Aplicando MVC podemos reaproveitar o mesmo dado para múltiplas visualizações;
- Podemos reaproveitar o comportamento (eventos) da solução;
- É um “pattern” de arquitetura, criado há muito tempo. Pode ser aplicado em qualquer linguagem, mais facilmente com OOP.

Model-view-controller

- 115.000 resultados na busca sobre framework MVC no google
- Struts, WebWorks, Spring, PicoContainer são exemplos de frameworks J2EE
- Você ainda não fez um framework MVC?
- A modinha dos jovenzitos do Inversion of Control

View Helper

- **Definição**: simplifica a “renderização” de objetos em views com formatação.
- Uma Custom Tag pode representar um View Helper;
- Uma simples classe convencional com métodos estáticos também;
- Exemplo no JAREF

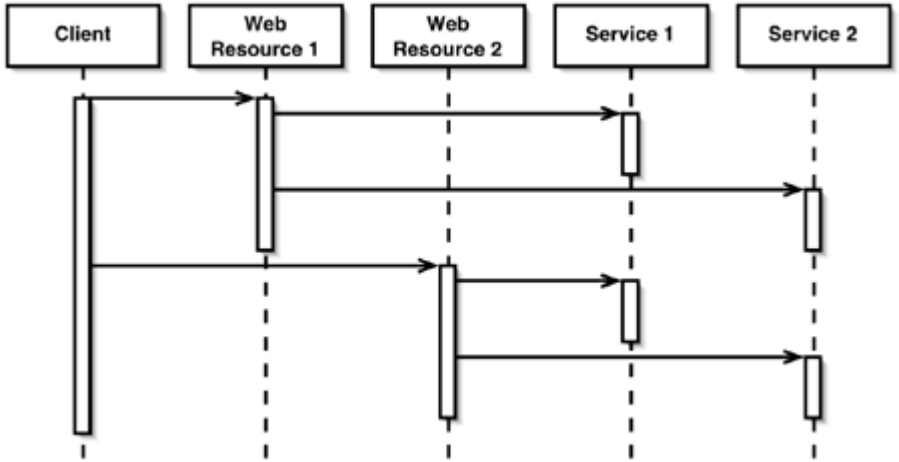
Intercepting Filter

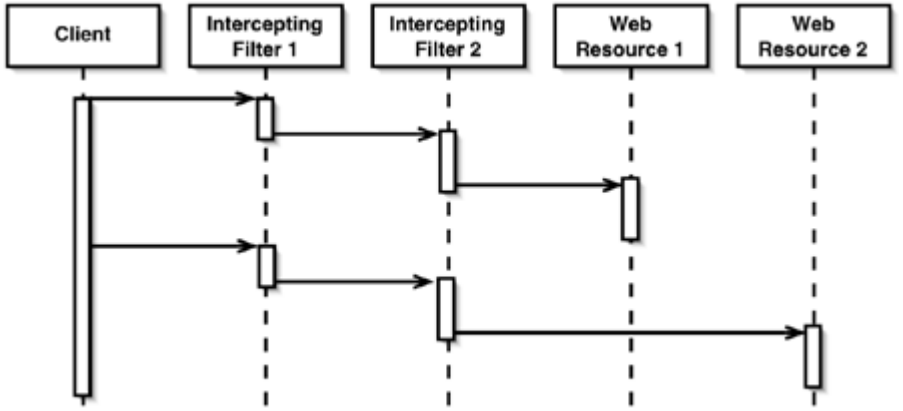
- **Definição**: forma para executar pré e pós processamento em requests da solução

- Um Servlet Filter é um exemplo de implementação de Intercepting Filter para interceptar requests no Web Container;

- **Ainda** não temos na espec J2EE intercepting filter para EJB's

Intercepting Filter

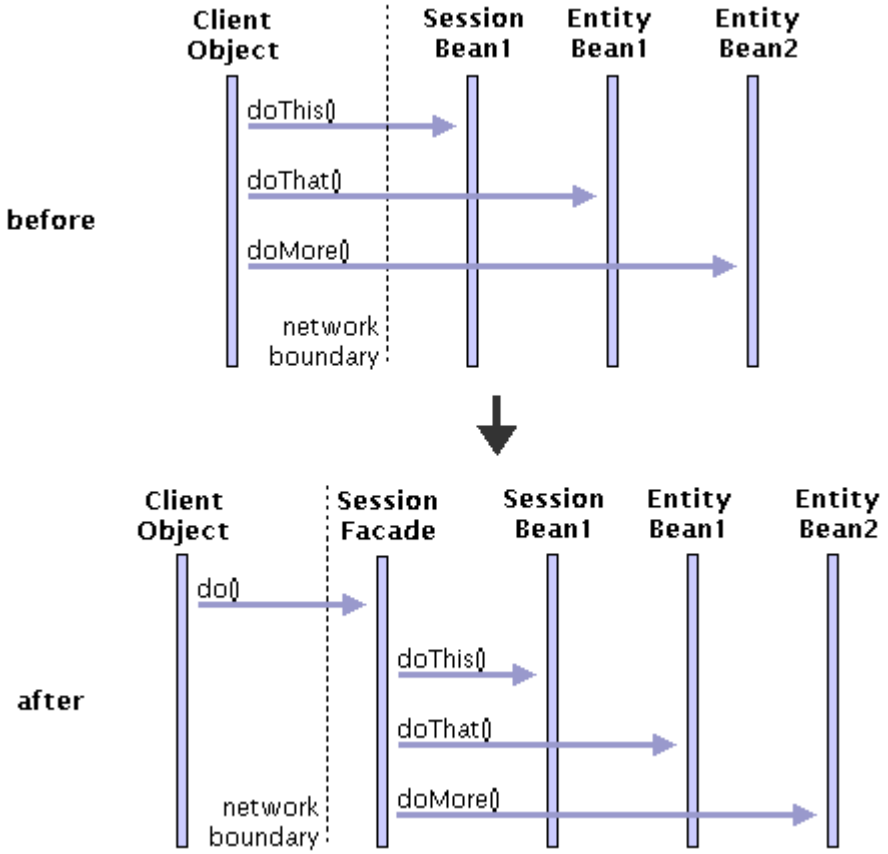




Session Façade

- **Definição**: muitos processos em servidores envolvem a manipulação de diversas business classes. O Session Façade cria uma fachada simplificada para representar um processo de negócio complexo.

- Exemplo JAREF



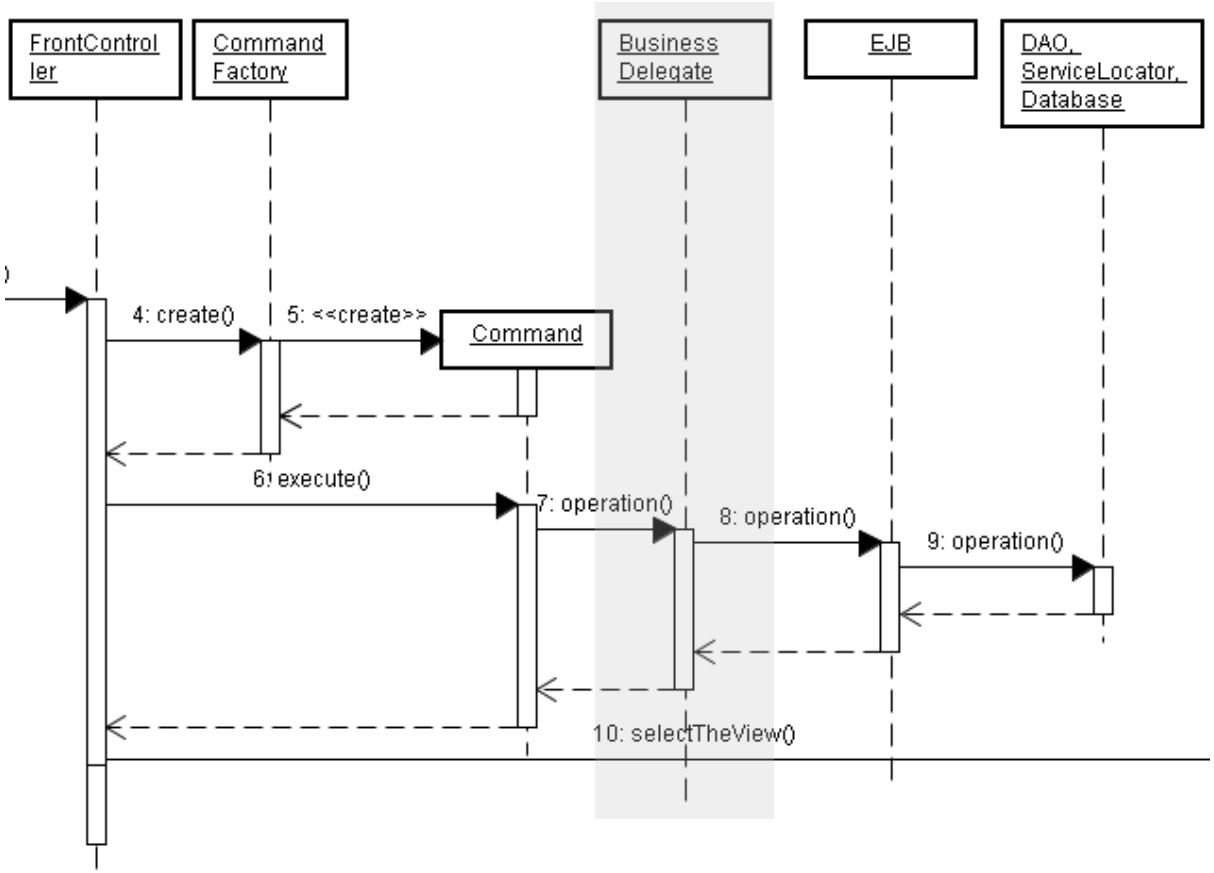
Business Delegate

- **Definição**: em aplicações distribuídas, o acesso remoto / local a EJB's via JNDI Naming Service e tratamento de erros pode se tornar complexo à medida que o projeto cresce.

- **Solução**: criar uma classe intermediária para acessar os EJB's que contempla as regras de nomes de componentes para lookups, propriedades do servidor J2EE, tratamento de exceptions, etc.;

Business Delegate

- **No JAREF....**



Outros...

- **Value List Handler:** provê uma forma eficiente para percorrer e interagir com grande quantidade de dados entre camadas;

- **Fast Lane Reader:** provê uma forma eficiente para acessar dados somente leitura para apresentação ou exportação do dado. Na prática o pattern recomenda utilizar EJBs com DAO e código SQL no lugar de Entity Beans.

- **Composite Entity**: uma forma de relacionar entitidades que são compostas por outras entidades. Implementado no CMP 2.0

J2EE Patterns Catalog

Each pattern in this catalog includes sample code from Java™ BluePrints reference applications such as the Java Pet Sto

Pattern Name	Description
Business Delegate [ACM01]	Reduce coupling between Web and Enterprise JavaBeans™ tiers
Composite Entity [ACM01]	Model a network of related business entities
Composite View [ACM01]	Separately manage layout and content of multiple composed views
Data Access Object (DAO) [ACM01]	Abstract and encapsulate data access mechanisms
Fast Lane Reader	Improve read performance of tabular data
Front Controller [ACM01]	Centralize application request processing
Intercepting Filter [ACM01]	Pre- and post-process application requests
Model-View-Controller	Decouple data representation, application behavior, and presentation
Service Locator [ACM01]	Simplify client access to enterprise business services
Session Facade [ACM01]	Coordinate operations between multiple business objects in a workflow
Transfer Object [ACM01]	Transfer business data between tiers
Value List Handler [ACM01]	Efficiently iterate a virtual list
View Helper [ACM01]	Simplify access to model state and data access logic

Agenda

- Introdução básica
- Patterns e certificações Sun
- GoF patterns: criação
- GoF patterns: comportamento
- GoF patterns: estrutura
- J2EE patterns
- Conclusões

Conclusões

- Os patterns J2EE são poucos e fáceis de entender;
- Utilizando patterns você cria soluções padronizadas, facilitando a troca de programadores;
- Você pode criar seus próprios patterns;
- O site theserverside.com contém vários patterns fora do catálogo J2EE e GoF;