UNIVERSIDADE CEUMA

Padrões de codificação Java (JSF)

Versão 1.2

17/06/2013

Este é um modelo de codificação para desenvolvimento Java existente no Grupo Ceuma de Educação. Aqui encontram-se os padrões utilizados por equipes internas e definições de como terceiros devem implementar seus serviços prestados.

Sumário

Estruturas de Pacotes

1 Pacotes

1.1 Estrutura

br.sigaweb

|_ **dao**: classes responsáveis por acesso a dados independente da fonte dados utilizada, por exemplo, EmpresaDAO para acessar dados referentes a entidade empresa.

|_ **dominio**: classes responsáveis por representar as entidades envolvidas no problema em questão, por exemplo, a classe Empresa no sistema.

|_ **excecao**: classes que representam exceções de negócio da aplicação em questão, elas herdam da exceção de negócio mãe (NegocioException), por exemplo exemplo, EmpresaNaoCadastradaException.

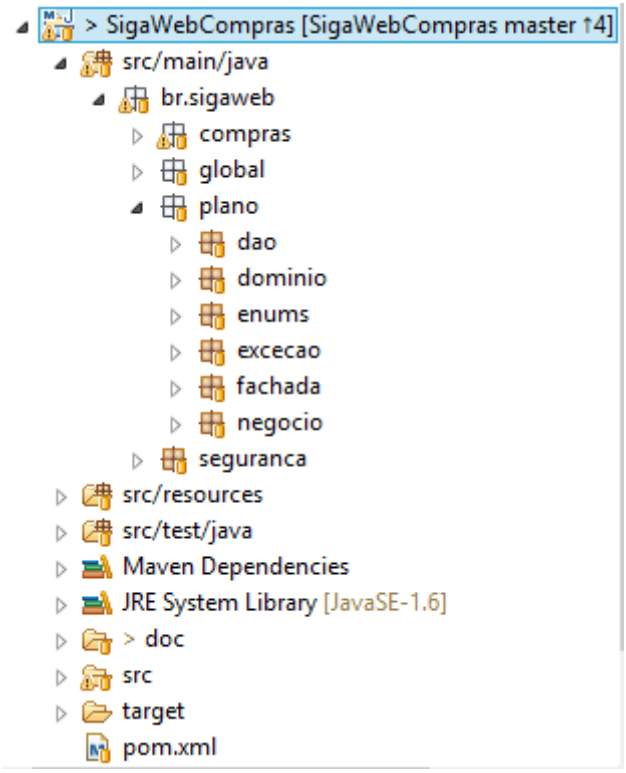
|_ **fachada**: classes que fornecem uma forma centralizada de comunicação entre a camada de negócio e a camada de visão, desta forma toda chamada a controle passa pela fachada, por exemplo, a classe EmpresaBean para realizar a persistência de uma empresa chama a ComprasFachada, por exemplo, para que ela delegue esta tarefa para a classe de negócio responsável.

|_ **negocio**: classes que representam regras de negócio que não pertencem a nenhuma classe do domínio. Por exemplo: a classe EmpresaNegocio contém as regras referentes a empresa que não podem ser implementadas dentro da classe Empresa. As classes de negócio fornecem uma maneira de acessar as funcionalidades dos dao's pelas camadas superiores.

|_ **visao**: classes que representam os beans do JSF. Normalmente para cada página temos um bean JSF correspondente. Por exemplo: a página de empresa.xhtml tem um bean chamado EmpresaBean

que realiza as operações invocadas na página.

A figura abaixo resume a estrutura de pacotes mencionada acima.



2 Nomenclatura

Os nome dos pacotes dos sistemas seguem a mesma convenção de nomes adotada pela sun (oracle), ou seja, a de domínio invertido. No caso de sistemas pertencentes ao SigaWeb fica, por exemplo: br.sigaweb.compras para o sistema SigaWebCompras.

2.1 Classes

2.1.1 DAO

A classe responsável por acessar dados da entidade empresa recebe o nome EmpresaDAO.

2.1.2 Exceção

A classe responsável por lançar uma exceção de empresa não

cadastrada recebe o nome EmpresaNaoCadastradaException.

2.1.3 Fachada

A fachada do sistema de contabilidade recebe o nome ComprasFachada.

2.1.4 Negocio

A classe responsável por conter as regras de negócio da classe empresa, que não podem ser implementadas na mesma, recebe o nome EmpresaNegocio.

2.2 Visão

O bean responsável por manter os dados de uma empresa recebe o nome de EmpresaBean.

2.3 Teste

As classes de teste seguem o mesmo padrão que outros tipos de classe com uma pequena modificação no seu nome, por exemplo, a classe responsável por testar a classe EmpresaNegocio se chama EmpresaNegocioTest (sem o “e” mesmo por causa do padrão do maven).

3 Métodos

3.1 DAO e Negocio (Padrão de CRUD)

- O método para atualizar uma empresa recebe o nome de **atualizar**.
- O método para excluir uma empresa recebe o nome de **excluir**.
- O método para recuperar uma empresa recebe o nome de **recuperar**.
- O método para recuperar várias empresas recebe o nome **recuperar**.

- O método para salvar uma empresa recebe o nome de **salvar**.
- O método para salvar ou atualizar uma empresa recebe o nome de **salvarOuAtualizar**.

3.2 Fachada (Padrão de CRUD)

- O método para atualizar uma empresa recebe o nome de **atualizarEmpresa**.
- O método para excluir uma empresa recebe o nome de **excluirEmpresa**.
- O método para recuperar uma empresa recebe o nome de **recuperarEmpresa**.
- O método para recuperar uma várias empresas recebe o nome **recuperarEmpresa**.
- O método para salvar uma empresa recebe o nome de **salvarEmpresa**.
- O método para salvar ou atualizar uma empresa recebe o nome de **salvarOuAtualizarEmpresa**.

3.3 Atributos

Lower CamelCase.

Exemplo:

- **int** numeroFax;
- **String** cargo;
- **int** codigoSetor;

3.4 Constantes

Maiúsculas e, caso sejam palavras compostas, devem ser separadas por underline. Por exemplo:

- MAXIMO
- CODIGO_SERVICO_TESTE

4 Arquivos de Configuração

4.1 Spring

- O arquivo que contém as definições dos ***data sources*** do sistema para base de desenvolvimento do ceuma recebe o nome de **NomeDoProjeto-Context-Desenvolvimento-Ceuma.xml**. No sistema SigaWebCompras, por exemplo, o nome do arquivo de configuração do spring, para acessar a base de desenvolvimento, é **SigaWebCompras -Context-Desenvolvimento-Ceuma.xml**.
- O arquivo que contem as definições dos ***beans*** do spring (não confunda com os managed beans do JSF) de um sistema recebe o nome de **NomeDoProjeto-Context-Beans.xml**. No sistema SigaWebCompras, por exemplo, o nome do arquivo de configuração de definições dos bean é **SigaWebCompras -Context-Beans.xml.**

1.1.2 Log4j

- O ***appender*** de para realizar o log em um arquivo do sistema de compras, por exemplo, recebe o nome COMPRAS e os arquivos devem ser salvos em uma pasta no servidor de aplicação com o nome **/var/log/sistemas/compras/compras.log.** E o padrão de data utilizado é dd-MM-yyyy.

5 Arquitetura

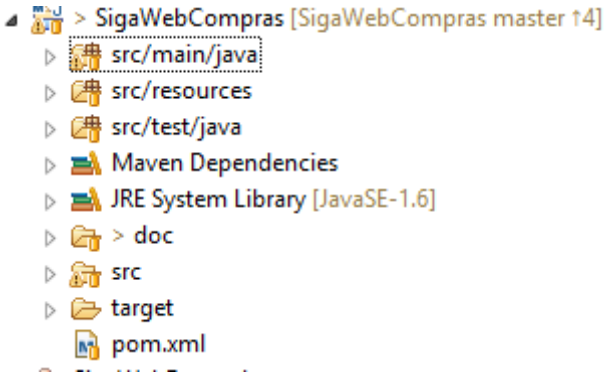
Os projetos Java seguem a estruturação MVC, isso implica desde o pacote de classes, até a arquitetura de configuração do projeto. Os projetos são criados com o framework **Maven** que segue a seguinte estrutura:

- **src/main/java**: Diretório que contém pacotes, classes, interfaces e enums.
- **src/resources**: Diretório que contém folders com seus respectivos

arquivos de configuração.

- **src/test/java**: Diretório que contém pacotes e classes referentes aos testes unitários.
- **src/main/webapp** (META-INF**;** WEB-INF): Diretório que contém páginas web.
- **pom.xml**: Arquivo raiz do maven, responsável por gerenciar as dependências do projeto.

O modelo descrito pode ser observado na imagem abaixo:



6 Hospedagem e Instalação

Para as aplicações é recomendável que possam ser hospedadas nos seguintes servers e/ou containers:

- Tomcat a partir da versão 6.0.37;
- JBoss a partir da versão 7;

7 Critérios de aceitação e controle de qualidade

Somente serão aceitos projetos que contemplem testes de unidade e mensurados conforme tabela abaixo:

Views	Testes de interface
DAO	Testes de persistência de dados
Negócio	Testes de regra de negócio

Os testes referentes a Fachada, DAO e Negócio deverão ser executados com o **JUnit** e passando todos, os resultados serão avaliados com o plugin **Emma** para verificação da cobertura de testes.

Os testes nas Views serão realizados com **Selenium WebDriver** e a cobertura aferida com o Emma.

Não serão consideradas aplicações que não tiverem seu projeto de testes, e que não possam ser aferidas pelos testes de unidade, integração e regressão.

8 Frameworks Aceitáveis para Desenvolvimento Java WEB

As tecnologias abaixo configuram os recursos que podem ser utilizados e tidos como aceitáveis para uso:

- **Maven 3**: Arquiteto da Aplicação; (**Obrigatório**)
- **Spring 3**: Integrador de Frameworks e gerenciador; (**Obrigatório**)
- **Hibernate 3**: Ferramenta para persistência de dados;
- **Mybatis 3**: Ferramenta para persistência de dados;
- **Facelets**: Responsável pela reutilização de layouts;
- **Tiles 2**: Responsável pela reutilização de layouts;
- **JSF 2**: Ferramenta para controle da visão;
- **Vraptor 3.5**: Ferramenta para controle da visão;
- **JSTL 1.2**: Ferramenta para melhorar a utilização de recursos da visão;
- **Primefaces 3.5**: Ferramenta de componentes e recursos visuais;
- **RichFaces 4**: Ferramenta de componentes e recursos visuais;

  As configurações possíveis são:

  - Maven + Spring + Hibernate (ou Mybatis) + JSF + PrimeFaces (ou RichFaces) + Facelets;
  - Maven + Spring + Hibernate (ou Mybatis) + Vraptor + JSTL + Tiles;