|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sistema** | | Multicalculo | | | |
| **Controle de Versão** | | | | | |
| **Versão** | **Controle** | | **Data** | **Razões para alteração** | **Responsável** |
| 01 | 00 | | 05/06/2012 | Criação | Adriano Santos |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |
|  |  | |  |  |  |

[1 Objetivo 5](#_Toc403640115)

[2 Premissas 5](#_Toc403640116)

[3 Visão Geral 6](#_Toc403640117)

[4 Diretrizes para Desenvolvimento 7](#_Toc403640118)

[4.1 Camada de Modelo 7](#_Toc403640119)

[4.1.1 Atributos de classe 7](#_Toc403640120)

[4.1.1.1 Considerações sobre parâmetros globais aplicação 7](#_Toc403640121)

[4.1.1.2 Utilização desnecessária de classe Wrapper 8](#_Toc403640122)

[4.1.2 Enums 8](#_Toc403640123)

[4.1.2.1 Utilização dos atributos 8](#_Toc403640124)

[4.1.2.2 Atributos e métodos de acesso 9](#_Toc403640125)

[4.1.2.3 Listas 9](#_Toc403640126)

[4.1.3 ORM (Object Relational Mapping) 11](#_Toc403640127)

[4.2 Camada de Apresentação 14](#_Toc403640128)

[4.2.1 JSTL 14](#_Toc403640129)

[4.2.2 Injeção de Dependências JSF 14](#_Toc403640130)

[4.2.3 Arquivos de propriedades e bundles 14](#_Toc403640131)

[4.2.4 Formulários 16](#_Toc403640132)

[4.2.4.1 Não utilização de flags 18](#_Toc403640133)

[4.2.4.2 Padrão para campos: Label / Campo / Tooltip 21](#_Toc403640134)

[4.2.4.3 Máscaras 22](#_Toc403640135)

[4.2.4.4 Botões 22](#_Toc403640136)

[4.2.4.5 Binding com o Controller 23](#_Toc403640137)

[4.2.4.6 Javascript 24](#_Toc403640138)

[4.2.4.7 Validação 25](#_Toc403640139)

[4.2.4.8 Mensagens 26](#_Toc403640140)

[4.2.4.9 Utilização de Ajax 26](#_Toc403640141)

[4.2.4.10 Fila Global para Ajax 27](#_Toc403640142)

[4.2.4.11 Utilização de a4j:pool 27](#_Toc403640143)

[4.2.4.12 Utilização de a4j:queue 28](#_Toc403640144)

[4.2.5 Navegação 28](#_Toc403640145)

[4.2.5.1 Menus 28](#_Toc403640146)

[4.2.5.2 Abas 28](#_Toc403640147)

[4.3 Camada de Serviço 29](#_Toc403640148)

[4.3.1 Criando um serviço 29](#_Toc403640149)

[4.3.2 Persistência 31](#_Toc403640150)

[4.3.2.1 Criando um DAO 32](#_Toc403640151)

[4.3.2.2 Efetuando cópia de entidades 32](#_Toc403640152)

[4.3.2.3 Criando buscas em entidades (Hibernate) 33](#_Toc403640153)

[4.3.3 O que é a Unidade de Trabalho 34](#_Toc403640154)

[4.3.3.1 Utilizando a Unidade de Trabalho 35](#_Toc403640155)

[4.3.3.2 Rastreando execução da unidade de trabalho 36](#_Toc403640156)

[4.3.4 Padrão de mensagens 36](#_Toc403640157)

[4.3.4.1 Tipos de mensagens 36](#_Toc403640158)

[4.3.4.2 Definição das mensagens 37](#_Toc403640159)

[4.3.4.3 Wrapper para mensagens por tipo 38](#_Toc403640160)

[4.3.4.4 Exemplo de implementação no serviço 38](#_Toc403640161)

[4.3.4.5 Wrapper para mensagens na Web 39](#_Toc403640162)

[4.3.5 Arquivos de configuração 39](#_Toc403640163)

[5 Implementação de exemplo 39](#_Toc403640164)

[5.1 Implementando uma aba 40](#_Toc403640165)

[5.1.1 Navegação 40](#_Toc403640166)

[5.2 Eventos 40](#_Toc403640167)

[5.2.1 Interface Listener 41](#_Toc403640168)

[5.2.2 Source 41](#_Toc403640169)

[5.2.2.1 O Announcer 41](#_Toc403640170)

[5.2.2.2 Exemplo 41](#_Toc403640171)

[5.2.3 Listener 42](#_Toc403640172)

[5.3 Cenários de exemplo 42](#_Toc403640173)

[5.3.1.1 Mensagens de alerta entre abas 42](#_Toc403640174)

[5.3.1.2 Mensagem de erro que impede continuação do fluxo de navegação 43](#_Toc403640175)

[5.3.1.3 Mensagem de validação que impede continuação do fluxo de navegação 43](#_Toc403640176)

[5.3.1.4 Mensagem de alerta que muda o fluxo de navegação 44](#_Toc403640177)

[5.3.1.5 ValueChangeListener + reRender: Em outra aba a partir de campo alterado 45](#_Toc403640178)

[5.3.1.6 A4j:support: Tratamento javascript a partir de processamento de um campo via Ajax. 45](#_Toc403640179)

[6 Nomenclatura 46](#_Toc403640180)

[6.1 Nomeação de classes 46](#_Toc403640181)

[6.2 Nomeação de métodos 46](#_Toc403640182)

[6.2.1 DAO 46](#_Toc403640183)

[6.2.2 Service 46](#_Toc403640184)

[7 Empacotamento e Importação 46](#_Toc403640185)

[7.1 VO 46](#_Toc403640186)

[7.2 Classes utilitárias 46](#_Toc403640187)

[7.3 Serviços 46](#_Toc403640188)

[8 Chave de Controle de Projeto Piloto 47](#_Toc403640189)

[8.1 Visão Geral 47](#_Toc403640190)

[8.2 Diagrama de classes 48](#_Toc403640191)

[8.3 Verificação de Projetos Ativos 49](#_Toc403640192)

[9 Execuções eventuais e atualitabs. 50](#_Toc403640193)

[9.1 Abertura do chamado 50](#_Toc403640194)

[9.1.1 Regras e horários para abertura 50](#_Toc403640195)

[9.1.1.1 Atualitab 50](#_Toc403640196)

[9.1.1.2 Execução eventual 50](#_Toc403640197)

[9.2 Projeto para execução de atualitabs 51](#_Toc403640198)

[9.2.1 Binário (JAR) 51](#_Toc403640199)

[9.2.2 Arquivo de comandos (.sql) 51](#_Toc403640200)

[9.2.3 Shell script (.sh) 51](#_Toc403640201)

[9.2.3.1 Parametros na shell 51](#_Toc403640202)

[9.2.4 Arquivos de log 52](#_Toc403640203)

[10 Uso indiscriminado dos métodos get. 52](#_Toc403640204)

[10.1 Utilização dos métodos get/set. 52](#_Toc403640205)

[10.1.1 Uso indiscriminado dos métodos get/set. 52](#_Toc403640206)

[10.1.1.1 Cenário de uso 52](#_Toc403640207)

# Objetivo

Fornecer visão geral da arquitetura da aplicação, estabelecer e documentar um padrão para desenvolvimento atuando como um guia de referência para o desenvolvedor envolvido no sistema.

Este documento não substitui as documentações de arquitetura da corporação e de referência das tecnologias e frameworks utilizados.

# Premissas

Tecnologias envolvidas:

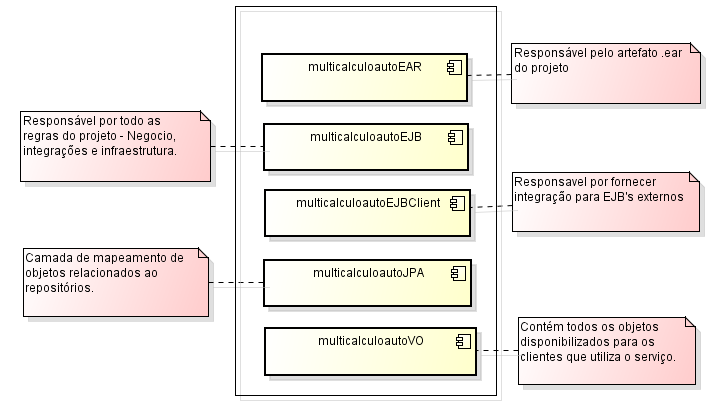
|  |  |  |  |
| --- | --- | --- | --- |
| Descrição | Versão | Nível de conhecimento desejado | Observação |
| Ambiente | JEE 6 | Intermediário |  |
| EJB | 2.1 e 3 | Intermediário | Integração e serviços |
| JMS | JEE 5 | Básico | Sistema possui integrações utilizando WebSphere MQ. |
| JUnit | 4.10 | Básico |  |
| Web Services | JEE 5 – JAXWS  Apache Axis 1.4 | Básico |  |
| JPA | 1.2 | Básico |  |
| Oracle | 11g | Básico |  |
| Maven | 2.0 | Básico |  |

Matriz tecnológica

|  |  |
| --- | --- |
| Descrição | Versão |
| Ambiente | JEE 6 |
| EJB | 2.1 e 3 |
| JMS | JEE 5 |
| JUnit | 4.10 |
| Spring Framework | 2.5.6 |
| Web Services | JEE 5 – JAXWS , Apache Axis 1.4 |
| Hibernate | 3.4.0.GA |
| Banco de Dados Oracle | 11g |
| Maven | 2.0 |

# Visão Geral

## Estrutura de módulos



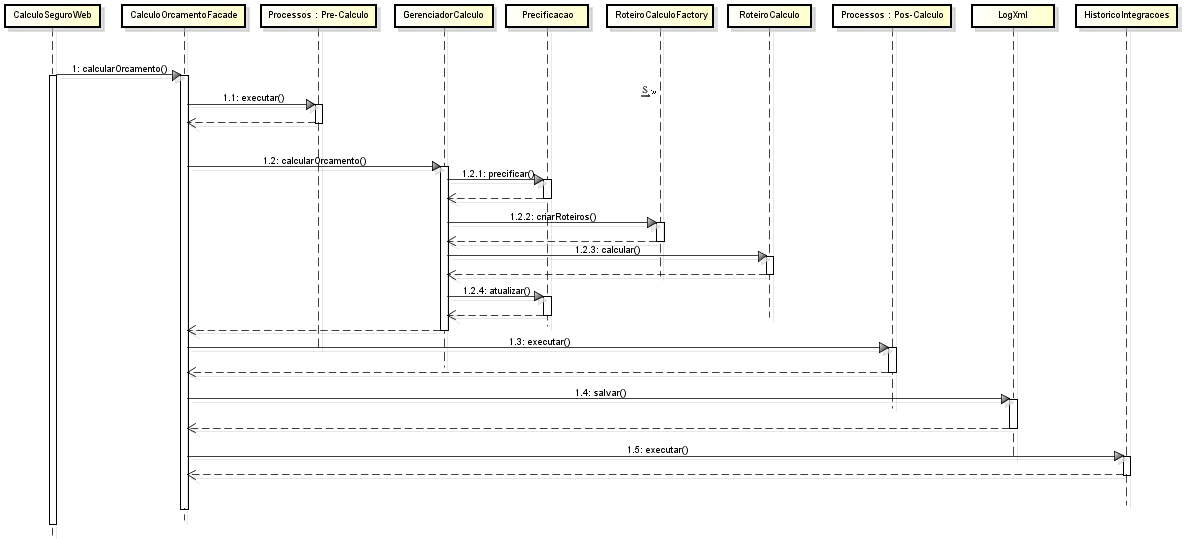
## Infraestrutura interna

### Estrutura em POJO’s (singleton).

Toda a Estrutura de regra e roteiro para execução do calculo foi elaborada em cima do padrão Singleton, pois foi consideramos a facilidade e simplicidade em adotar essa solução, tirando toda a complexidade de EJB’s ou outra solução de IOC.

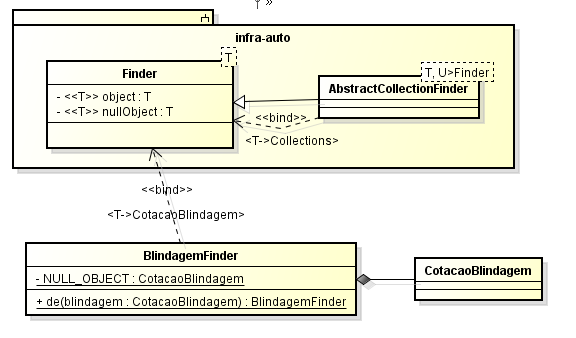
### Roteiro de Cálculo.

O termo roteiro de cálculo corresponde a regra de cálculo e seu fluxo para o devido calculo, para um entendimento macro do fluxo entre os objetos para o roteiro, segue um diagrama de sequencia:

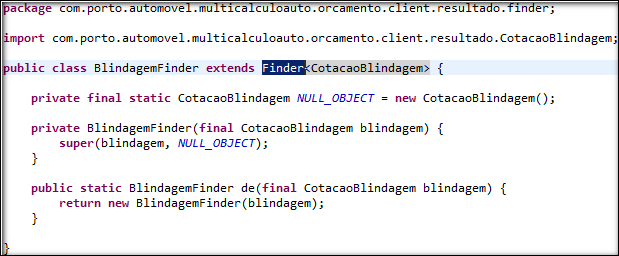


### Finders

Com a necessidade de navegar pela a estrutura complexa dos os objetos de domino da aplicação, foi criado o conceito de Finder, o mesmo é responsável pela navegação lembrando um Interator de Set, porém especifico para o objeto de domínio, para exemplificar segue :



Para criar um finder devemos, especializar a classe Finder, iniciar o NullObject deixando assim o novo finder null safe. A ligação entre os Finders é natural e necessária para utilizar o máximo dos benefícios dessa estrutura, podendo assim apresentar um alto nível de dependências entre os Finders



### Converters

### Processos

Para todas as ações que devem ser efetuadas em paralelo ao calculo indiferente se antes ou pós, as mesmas devem ser enquadradas como um processo no fluxo do multicalculo .

Todo o processo deve seguir a estrutura abaixo para que seja reaproveitada a estrutura criada:

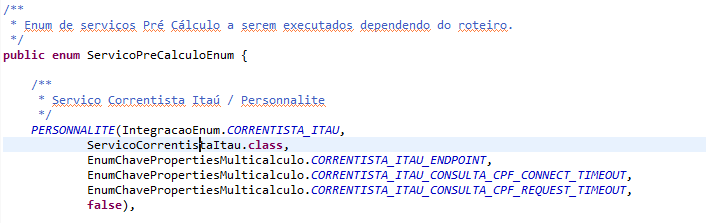
### Integrações

#### Assíncrona

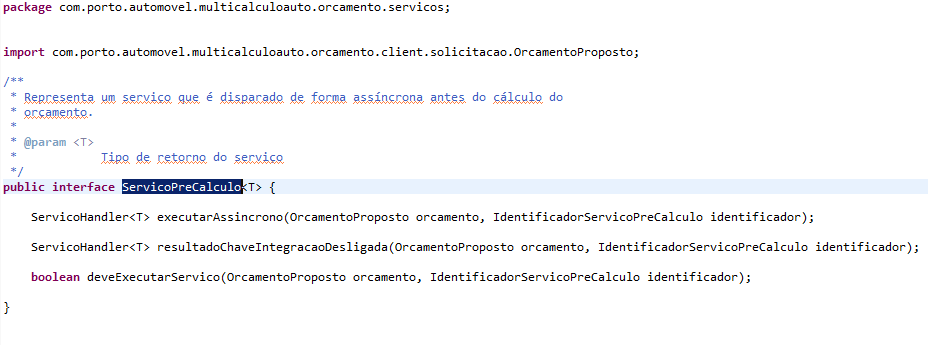
Considera-se que todas as integrações que na sejam a referente ao calculo são assíncronas pois são complemento ou requisitos para o calculo.

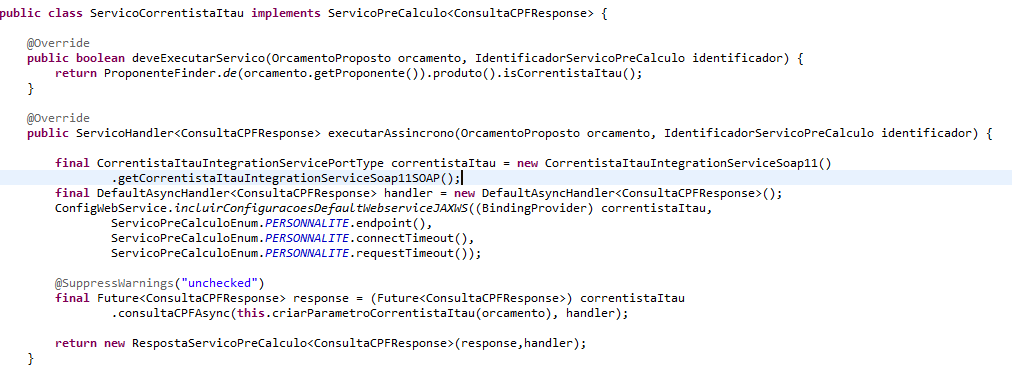
Uma integração assíncrona começa a ser representada de duas formas Pre ou Pós cálculo:

Para a integração que se enquadra devemos criar uma representação da mesma na Enum de Pré-cálculo como o exemplo abaixo.



A responsabilidade da execução referente à integração fica para a classe que implementa a ServioPreCalculo que de acordo com o exemplo acima é ServicoCorrentistaItau.java, a mesma é passada no construtor da Enum. .





#### Síncrona

**public** **enum** TipoEndossoEnum {

*NENHUM*(0), *INCLUSAO*(1, "Inclusão de Item"),

*SUBSTITUICAO*(2, "Substituição de Veículo");

**private** **final** String descricao;

**private** **final** Integer code;

No exemplo acima os atributos **descrição** e **code** não devem ser utilizados de forma individual numa classe. Um objeto deve conter uma referência para o enum e não os atributos separadamente. Um objeto deve ter como atributo algo que seja do tipo **TipoEndossoEnum** e não um atributo **String**(descricao)e/ou outro atributo **Integer**(code). Seguem exemplos:

Exemplos incorretos:

/\*\*

\* VO para Dados da Tela de Busca/Carga de Endosso.

\*/

**public** **class** SolicitacaoEndossoVO **implements** Serializable {

**private** Integer tipoEndosso;

**private** String descricaoTipoEndosso;

/\*\*

\* VO para Dados da Tela de Busca/Carga de Endosso.

\*/

**public** **class** SolicitacaoEndossoVO **implements** Serializable {

**private** Integer codigotipoEndosso;

No exemplo de utilização correta, ambas as utilizações devem referenciar diretamente o **TipoEndossoEnum**:

/\*\*

\* VO para Dados da Tela de Busca/Carga de Endosso.

\*/

**public** **class** SolicitacaoEndossoVO **implements** Serializable {

**private** TipoEndossoEnum tipoEndosso;

Então se for necessário obter o atributo **code** ou o atributo **descrição**. Devem ser usados os métodos de acesso (*get*) criados no enum. Exemplo: ***tipoEndosso.getCode()*** ou ***tipoEndosso.getDescricao().***

Isso evita que objetos desnecessários sejam criados no *pool* de Strings, ou na área permanente (*PermGen*) do *heap.*

Outro benefício é que o desenvolvedor ou analista sabe exatamente os possíveis valores do atributo, evitando erros ou atribuições de valores não relacionados com a lógica.

#### Atributos e métodos de acesso

Os enums que tiverem atributos estes devem ser definidos com o modificador de acesso ***final***. Pois desta forma evita-se que um desenvolvedor, intencionalmente ou não, altere o valor do mesmo utilizando os métodos ***set***.

Conclue-se então, que atributos em enums, devem ser ***final*** e não possuir método ***set.***

#### Listas

Nota-se que pela complexidade do negócio, em muitas situações são necessários enums complexos com atributos customizados, por exemplo:

**public** **enum** TipoDocumentoIdentificacaoPessoaEnum {

*RG*(1, "R.G."),

*RNE*(2, "R.N.E.")

...

**private** **final** Integer codigo;

**private** **final** String descricao;

...

**public** **static** List<Integer> *LISTA\_DOCUMENTOS\_IDENTIFICACAO*;

**public** **static** List<Integer> *LISTA\_DOCUMENTOS\_IDENTIFICACAO\_CARTAO*;

...

}

Observe que os atributos públicos citados neste enum (LISTA\_DOCUMENTO\_IDENTIFICACAO E LISTA\_DOCUMENTO\_IDENTIFICACAO\_CARTAO) serão instâncias de classe que implementem List provavelmente será utilizado ArrayList.

Um cuidado que deve-se ter com este tipo de implementação é que uma instância da classe ArrayList é mutável (permite alterações em seu conteúdo). Sendo assim, outro trecho de código por falta de atenção e/ou conhecimento pode alterar o conteúdo destes atributos que devem ser fixos.

* Solução proposta: O atributo da lista deve ter seu conteúdo iniciado em um bloco estático e receber uma referência de uma lista que seja imutável, como por exemplo utilizar ***Arrays.asList(…)***
  + Exemplo correto:

**public** **enum** TipoDocumentoIdentificacaoPessoaEnum {

...

**public** **static** List<Integer> *LISTA\_DOCUMENTOS\_IDENTIFICACAO*;

*...*

**static** {

*LISTA\_DOCUMENTOS\_IDENTIFICACAO* = Arrays.*asList*(

**new** Integer[]{

*RG*.getCodigo(),

*RNE*.getCodigo(),

*CLASSE*.getCodigo(),

*RG\_CARTAO*.getCodigo(),

*RNE\_CARTAO*.getCodigo(),

*CLASSE\_CARTAO*.getCodigo()});

...

}

}

* + Exemplo incorreto:

**public** **enum** TipoDocumentoIdentificacaoPessoaEnum {

...

**public** **static** List<Integer> *LISTA\_DOCUMENTOS\_IDENTIFICACAO*;

*...*

**static** {

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*RG*.getCodigo());

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*RNE*.getCodigo());

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*CLASSE*.getCodigo());

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*RG\_CARTAO*.getCodigo());

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*RNE\_CARTAO*.getCodigo());

*LISTA\_DOCUMENTOS\_IDENTIFICACAO*.add(*CLASSE\_CARTAO*.getCodigo());

...

}

}

### Entity

No desenvolvimento de aplicações utilizando o Hibernate como Framework ORM, não raro há problemas com implementações falhas nas ORMs. Por isso, com o objetivo de padronizar as ORMs do PPWEB e reduzir problemas de performance, foram estabelecidos os seguintes padrões:

* + Mapeamento

Exemplo: Tabela APOMSEGDOC – com PK SEGDOCCOD, deve estar mapeado no arquivo hbm.xml (DocumentoSeguro.hbm.xml) da seguinte forma:

<id name=*"codigoDocumento"*

type=*"java.lang.Integer"*

column=*"SEGDOCCOD"* >

Para este mapeamento foi criada a classe JAVA contida em DocumentoSeguro.java:

/\*\*

\* documento seguro automovel

\*/

**public** **class** DocumentoSeguro **implements** Serializable {

/\*\* identifier field \*/

**private** Integer codigoDocumento;

...

}

* + Equals e HashCode
    - Equals

O método ***equals(…)*** é utilizado pelo Java para definir se um objeto é semelhante ao outro. Quando este objeto é uma entidade que será gerenciada pelo Hibernate devem ser utilizados para determinar a igualdade apenas os atributos que estiverem definidos como *ID* da ORM (deverá ser o mesmo *ID – PK* da tabela do banco de dados).

Deve ser utilizado ***instanceof*** ao invés de ***equals*** da classe.

Não devem ser utilizados *EqualsBuilders.*

Exemplo:

**public** **class** DocumentoSeguro **implements** Serializable {

...

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (!(obj **instanceof** DocumentoSeguro))

**return** **false**;

DocumentoSeguro other = (DocumentoSeguro) obj;

**if** (codigoDocumento == **null**) {

**if** (other.codigoDocumento != **null**)

**return** **false**;

} **else** **if** (!codigoDocumento.equals(other.codigoDocumento))

**return** **false**;

**return** **true**;

}

...

}

* + - HashCode

Para a implementação do ***hashcode(…)*** deve ser utilizado somente os atributos que estiverem definidos como *ID* da ORM (deverá ser o mesmo *ID – PK* da tabela do banco de dados).

Deve ser utilizada a implementação padrão do Eclipse.

Não devem ser utilizados *HashCodeBuilders.*

Exemplo:

* + Observações gerais

**public** **class** DocumentoSeguro **implements** Serializable {

...

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + ((codigoDocumento == **null**) ? 0 : codigoDocumento.hashCode());

**return** result;

}

...

}

As ORMS devem conter apenas métodos de acesso (getters e setters)

Nenhum método ***get*** deve possuir comportamento ou validação. Exemplos incorretos:

**public** **class** OperacaoEmpresa **implements** Serializable {

...

**public** String getNomeEmpresaOperacaoNovo() {

String nomeEmpresa = "";

**if**(getComp\_id().getCodigoEmpresaOperacao() != **null** && getComp\_id().getCodigoEmpresaOperacao().equals(0)){

nomeEmpresa = **this**.nomeEmpresaOperacao;

}**else**{

**if**(**this**.cadastroPlanoOperacaoEmpresa != **null**){

nomeEmpresa = **this**.cadastroPlanoOperacaoEmpresa.getNomeEmpresaOperacao();

}

}

**return** nomeEmpresa;

}

**public** **void** setNomeEmpresaOperacaoNovo(String nomeEmpresaOperacaoNovo) {

**if**(getComp\_id().getCodigoEmpresaOperacao() != **null**){

**if**(getComp\_id().getCodigoEmpresaOperacao().equals(0)){

**this**.nomeEmpresaOperacao = nomeEmpresaOperacaoNovo;

}**else** **if**(getComp\_id().getCodigoEmpresaOperacao().equals(1)){

**this**.nomeEmpresaOperacao = nomeEmpresaOperacaoNovo;

}**else** {

**if**(**this**.cadastroPlanoOperacaoEmpresa != **null**){ **this**.cadastroPlanoOperacaoEmpresa.setNomeEmpresaOperacao(nomeEmpresaOperacaoNovo);

**this**.nomeEmpresaOperacao = nomeEmpresaOperacaoNovo;

}

}

}**else**{

**this**.nomeEmpresaOperacao = nomeEmpresaOperacaoNovo;

}

}

...

}

## Camada de Apresentação

### JSTL

O uso de JSTL não é permitido na aplicação, qualquer verificação deve ser realizada pelos controllers gerenciados pelo JSF e qualquer apresentação deve ser realizada a partir de componentes xhtml ou JSF.

### Injeção de Dependências JSF

Para efetuar a injeção de dependências de beans (sejam do Spring ou do Faces) em um managed bean do Faces deve ser feito pelo arquivo faces-managed-beans.xml. Conforme exemplo abaixo verificamos a injeção de um bean do Spring (cadastroSusepFacade) em um managed bean do Faces (modalGerarPropostaBean):

<managed-bean>

<managed-bean-name>modalGerarPropostaBean</managed-bean-name>

<managed-bean-class>com.porto.portoprint.automovel.bean.ModalGerarPropostaBean</managed-bean-class>

<managed-bean-scope>session</managed-bean-scope>

<managed-property>

<property-name>cadastroSusepFacade</property-name>

<value>#{cadastroSusepFacade}</value>

</managed-property>

...

</managed-bean>

### Arquivos de propriedades e bundles

As inclusões e alterações de mensagens ou labels (fixos ou dinâmicos) em arquivos de propriedades (\*.properties) ou diretamente em um javascript (\*.js) devem considerar:

* + Utilização apenas de caracteres simples;
  + Para inserção de caracteres especiais ou alfa numéricos acentuados (á, é, í, ó, ú, ê...) utilizar os códigos Unicode com escape Javascript.
  + Exemplo de frases e sentenças incorretas:
    - Não deve conter mais de 01 (um) veículo na proposta do auto.
    - ¨´ºª#$%&()\*+,/:;<=>?@[\\\\\\\\]\\\\\\\'^\_`|}~{!\\\"¹²³£¢¬-–¤¶§ĒÜÉĀÄĄÅĒŹĖČĻĪĢĘĘ«»¦±°•·€„…†‡ˆ‰Š‹Œ‘’“”—˜™š›ĪĻŠŅÓŌÕÖ×ŲŁŚŪÜŻĆŠÕ÷ŲÜŻ©

* + Exemplo das frases acima convertidas:
    - N\u00E3o deve conter mais de 01 (um) ve\u00EDculo na proposta do auto.
    - \u00A8\u00B4\u00BA\u00AA#$%&()\*+,/:;<=>?@[\\\\\\\\]\\\\\\\'^\_`|}~{!\\\"\u00B9\u00B2\u00B3\u00A3\u00A2\u00AC-\u2013\u00A4\u00B6\u00A7\u0112\u00DC\u00C9\u0100\u00C4\u0104\u00C5\u0112\u0179\u0116\u010C\u013B\u012A\u0122\u0118\u0118\uF8FC\u00AB\u00BB\u00A6\u00B1\u00B0\u2022\u00B7\u20AC\u201E\u2026\u2020\u2021\u02C6\u2030\u0160\u2039\u0152\u2018\u2019\u201C\u201D\u2014\u02DC\u2122\u0161\u203A\u012A\u013B\u0160\u0145\u00D3\u014C\u00D5\u00D6\u00D7\u0172\u0141\u015A\u016A\u00DC\u017B\u0106\u0160\u00D5\u00F7\u0172\u00DC\u017B\u00A9

Para chamadas de algum arquivo de propriedades nas classes, evitar a utilização de constantes e chamadas a todo o momento algum valor do *properties*.

**public class VeiculoBean extends AbstractFormBean {**

**…**

**private** **static** **final** String *VEICULO\_PROPERTIES* = "properties.veiculo";

**private static final** String *MSG\_ERRO\_VALIDACAO\_ANO\_MODELO* = "veiculo.anoModeloInvalido";

**private static final** String *MENSAGEM\_NOME\_CORRETOR* = " mensagemNomeCorretor ";

**…**

**if**(**…**){

~~setMensagem~~(getMessage(*MSG\_ERRO\_VALIDACAO\_ANO\_MODELO*));

// OU

String msg = ~~getMessageResourceString~~(*MENSAGEM\_NOME\_CORRETOR*,

*VEICULO\_PROPERTIES*, **null**));

// OU

String msgOld = ~~getMessageResourceString~~("mensagemNomeCorretor",

*VEICULO\_PROPERTIES*, **null**));

}

**…**

**}**

Exemplos incorretos:

O exemplo abaixo mostra um *Enum*, que contem os códigos das mensagens e a leitura das mesmas, padronizando assim as chamadas do *properties*, evitando duplicações de constantes em vários pontos do código e também facilitando o retrabalho em futuras modificações.

Exemplo correto: *Enum* utilizando o message.properties:

**public** **enum** MessagesPropertiesEnum {

*ERRO\_TRANSMISSAO\_CONEXAO\_AZUL*("erro.transmissao.conexao.azul"),

*ERRO\_TRANSMISSAO\_OBTER\_DOCSEGAZUL*("erro.transmissao.obter.documentoseguroazul"),

*ERRO\_CADASTRAR\_USUARIO\_SENHA*("erro.cadastrar.senha.usuario");

**private** **static** **final** String *PROPERTIES\_CONF* = "properties.messages";

**private** **final** String key;

**private** **final** String value;

**private** MessagesPropertiesEnum(**final** String key){

**this**.key = key;

ResourceBundle resource = ResourceBundle.*getBundle*(*PROPERTIES\_CONF*);

**this**.value = resource.getString(key);

}

**public** String value() {

**return** value;

}

**public** String getKey() {

**return** key;

}

}

**public class PropostaVeiculoBean extends AbstractFormBean {**

**…**

**if**(**…**){

String mensagemErro = MessagesPropertiesEnum.*ERRO\_TRANSMISSAO\_CONEXAO\_AZUL.*value();

}

**…**

**}**

### Formulários



<h:form id=*"form\_id"* >

<!-- inicio mensagens formulario -->

<rich:messages id=*"message\_id"* layout=*"table"* errorLabelClass=*"errorMessage"* >

<f:facet name=*"errorMarker"*>

<h:graphicImage value=*"/images/ico-erro.gif"* alt=*"ico-erro"* />

</f:facet>

</rich:messages>

<table id=*"messageBoxCapaTable"* >

<tr>

<td>

<ul id=*"messageBoxCapa"* class=*"rich-messages"*></ul>

</td>

</tr>

</table>

<br class=*"clr"* />

<!-- fim mensagens formulario -->

<ui:include src=*"/orcamento/titulo\_secao.xhtml"* >

<ui:param name=*"titulo"* value="#{tela.tituloDaSecao}" />

</ui:include>

<div class=*"conteudo-sanfona-interna"*>

-- CAMPOS --

</div>

<!-- inicio botoes -->

<h:panelGroup >

<div class=*"f-right"*>

<div class=*"btCor1 col-lado-a-lado"*>

<!-- botao 1 -->

</div>

<div class=*"btCor4 col-lado-a-lado"*>

<!-- botao 2 -->

</div>

</div>

</h:panelGroup>

<!-- fim botoes -->

<script type=*"text/javascript"*>

// javascript, caso necessario

</script>

<br class=*"clr"* />

</h:form>

* Cada formulário deve possuir um ManagedBean controlador, caso possua comportamento implementado no cliente, deve possuir um objeto javascript para encapsular todos seus comportamentos. Sob qualquer hipótese não devem ser criadas funções ou variáveis javascript em contexto global!
* Os valores para os campos devem ser encapsulados em uma classe Java apenas com getters e setters .

Exemplo:

// Wrapper para o formulario:

**public** **class** MeusDados {

**private** String campo1;

**private** String campo2;

... getters/setters...

}

// ManagedBean – Controller JSF:

**public** **class** MeuFormularioController **extends** AbstractBaseBean {

**private** MeusDados dadosForm;

}

// Controller Javascript:

**var** MeuFormularioController = {

meuComportamento: **function**(){

},

MINHA\_CONSTANTE: 1,

minhaVariavel: **null**

};

#### Não utilização de flags

##### Controle de atributos e comportamento de campos por meio de flags

Alguns campos em tela tem comportamentos e atributos (renderização ou habilitação) controlados por regras de negócio ou dependentes do preenchimento de determinada informação em outro campo. Para estes não devem ser utilizadas **flags** para controlá-los, por exemplo, ***renderizarContratarCoberturaKitGas*** e ***desabilitaCampoRenovacaoSeguradora.***

Pontos contra:

* Dependência da aplicação executar determinado método para atualizá-la, podendo então o formulário estar com estado inconsistente (se em determinado cenário a flag não for atualizada);
* Complexidade de entendimento das condições de cada comportamento ou atributo;
* Classes maiores e mais complexas;

Utilização correta no XHTML:

<h:panelGroup id=*"kitGasSerie"* styleClass=*"div-mensagem-erro"*>

<div class=*"col-pequeno"* style=*"width: 130px;"*>

<label jsfc=*"h:outputLabel"* rendered=*"#{orcamentoVeiculoController.veiculoForm.renderizarContratarCoberturaKitGas}"* class=*"label"*>

Exemplo incorreto de implementação de POJO:

**public** **class** VeiculoForm {

**private** DadosComplementaresVeiculo dadosComplementaresVeiculo;

**private** **boolean** renderizarContratarCoberturaKitGas;

get, set...

}

Exemplo incorreto de implementação de regra de negócio para setar flag:

/\*\*

\* Carregar exibicao campo zero km.

\*/

**private** **void** carregarExibicaoCampoZeroKm() {

**boolean** desabilitaZeroKm = verificarPermissaoParaVeiculoZeroKm();

**if**(desabilitaZeroKm){

veiculoForm.getDadosVeiculo().setVeiculoZeroKm(Boolean.*FALSE*);

}

veiculoForm.getDadosVeiculo().setVeiculoZeroKmDisabled(desabilitaZeroKm);

**if**(veiculoForm.getDadosComplementaresVeiculo().getVeiculoKitGas() != **null**

&& veiculoForm.getDadosComplementaresVeiculo().getVeiculoKitGas()){

**if**(veiculoForm.getDadosComplementaresVeiculo().getValorCoberturaKitGas() != **null**){

veiculoForm.getDadosComplementaresVeiculo().setContratarCoberturaKitGas(Boolean.*TRUE*);

veiculoForm.setRenderizarValorCoberturaKitGas(Boolean.*TRUE*);

}**else**{

veiculoForm.getDadosComplementaresVeiculo().setContratarCoberturaKitGas(Boolean.*FALSE*);

}

**if** (!veiculoForm.isRenderizarKitGasSerie()) {

veiculoForm.setRenderizarContratarCoberturaKitGas(Boolean.*TRUE*);

}

}**else**{

veiculoForm.setRenderizarContratarCoberturaKitGas(Boolean.*FALSE*);

veiculoForm.getDadosComplementaresVeiculo().setContratarCoberturaKitGas(**null**);

**veiculoForm.setRenderizarValorCoberturaKitGas(Boolean.*FALSE*);**

veiculoForm.getDadosComplementaresVeiculo().setValorCoberturaKitGas(**null**);

}

}

Estes campos destacados com fundo vermelho não deverão existir neste ponto, segue abaixo a refatoração correta, deste caso. Notem que no XHTML não tem alteração visto que a assinatura do método ***get*** segue a convenção.

Pontos pró:

* Evitar invocar métodos para controlar flags;
* Comportamento de componentes de tela automaticamente sincronizados com dados do VO;
* Simplificação de código e diminuição de tamanho de classe;
* Evitar métodos redundantes ou atualização de flags em pontos descentralizados;

**public** **class** VeiculoForm {

**private** DadosComplementaresVeiculo dadosComplementaresVeiculo;

**//private** **boolean** renderizarContratarCoberturaKitGas; (exclusão de atributo)

**public** **boolean** isRenderizarContratarCoberturaKitGas() {

**boolean** retorno = Boolean.*FALSE*;

**if**(dadosComplementaresVeiculo.getVeiculoKitGas() != **null**

&& dadosComplementaresVeiculo.getVeiculoKitGas()){

**if**(dadosComplementaresVeiculo.getValorCoberturaKitGas() != **null**){

retorno = Boolean.*TRUE*;

}

**if** (!isRenderizarKitGasSerie()) {

retorno = Boolean.*TRUE*;

}

}

**return** retorno;

}

}

##### “Tradução/simplificação” de atributos com flags

Algumas vezes para simplificação de lógicas, ou utilização de atributos, criam-se novos atributos booleanos (flags) que representam determinada informação ou dados de um VO. Tais como o exemplo abaixo:

**public** **class** VeiculoForm {

**private** DadosVeiculo dadosVeiculo;

**private** **boolean** cambioAutomaticoSerie;

get, set...

}

A flag ***cambioAutomaticoSerie*** foi criada apenas para representar (traduzir/simplificar) um dados do veículo. Algo que já existe em ***DadosVeiculo***. Esta implementação gerou lógica de negócio duplicada, como vemos nas classes abaixo este mesmo atributo é setado em dos locais considerando as mesmas informações:

**public** **class** DadosComplementaresVeiculoMapper {

**public** DadosComplementaresVeiculo mapFrom(**final** DadosVeiculoOrcamento dadosVeiculoOrcamento,

**final** VeiculoForm form) {

**...**

**if** (dadosVeiculoOrcamento.getVersoesVeiculo() != **null**

&& dadosVeiculoOrcamento.getVersoesVeiculo().getTipoCambioVeiculo() == TipoCambioEnum.*AUTOMATICO*.getShortValue()) {

**form.setCambioAutomaticoSerie(true);**

}**else**{

**form.setCambioAutomaticoSerie(false);**

}

}

}

**public** **class** DadosComplementaresVeiculoMapper {

**public** **void** montarDescricaoVeiculo(){

...

**final** **int** tipoCambio = veiculoForm.getDadosVeiculo().getTipoCambioVeiculo();

**if** (tipoCambio == TipoCambioEnum.*MANUAL*.getValue()){

...

**veiculoForm.setCambioAutomaticoSerie(false);**

} **else** **if** (tipoCambio == TipoCambioEnum.*AUTOMATICO*.getValue()){

...

veiculoForm.setCambioAutomaticoSerie(**true**);

}

...

}

}

Para a correção deste caso deve ser removido o atributo “desnescessário” ***cambioAutomaticoSerie*** e ter a atribuição de valor da flag ser movida para o método ***get*** . Seguindo convenção. Conforme exemplo abaixo.

Os pontos pró são os mesmos do item 4.2.4.1.1.

**public** **class** VeiculoForm {

**private** DadosVeiculo dadosVeiculo;

//**private** **boolean** cambioAutomaticoSerie; (exclusão de atributo)

**public** **boolean** isCambioAutomaticoSerie() {

**boolean** retorno = **false**;

**final** **int** tipoCambio = dadosVeiculo.getTipoCambioVeiculo();

**if** (TipoCambioEnum.*AUTOMATICO*.getValue().equals(tipoCambio)){

retorno = **true**;

...

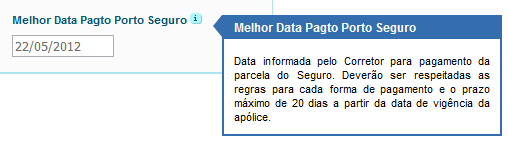
}

**return** retorno;

}

}

#### Padrão para campos: Label / Campo / Tooltip





<!-- 1o. PanelGroup (OBS: pode ser utilizado apenas uma DIV com a classe de tamanho Ex.: <div class="col-medio">)

Se faz necessária a utilização do PanelGroup apenas quando referenciado por algum evento JSF

(Ex.: utilização da propriedade rendered, binding com managed bean, etc)

-->

<h:panelGroup id=*"painelMelhorData"* layout=*"block"* class=*"col-medio data ultima-col div-mensagem-erro"* style=*"width: 190px;"* >

<!-- 2o. DIV com padrão para apresentação de mensagens de erro -->

<div class=*"div-mensagem-erro"*>

<!-- 3o. LABEL (Note a utilização de resource bundles para todos os textos em tela)-->

<b><h:outputLabel value="#{controleCapa.labelDataPagamento}" id=*"labelMelhorData"* styleClass=*"label f-left"* /></b>

<!-- 4o. Padrão para apresentação de erro -->

<h:graphicImage alt=*"ico-erro"* value=*"/images/ico-erro.gif"*

styleClass=*"mensagem-erro f-right"* style=*"margin-top: 1px; display: none;"* />

<rich:message for=*"melhorData"* showDetail=*"false"* showSummary=*"false"*>

<f:facet name=*"errorMarker"*>

<h:graphicImage alt=*"ico-erro"* value=*"/images/ico-erro.gif"*

styleClass=*"f-right"* style=*"margin-top: 1px; cursor: help;"* />

</f:facet>

</rich:message>

<!-- 5o. Tooltip -->

<div style=*"margin-right: -7px; margin-top: 1px"*>

<toolTipAjuda:ajuda campo=*"melhorData"* form=*"form\_capa"* titulo="#{controleCapa.labelDataPagamento}" descricao="#{capa.ajudaData}" />

</div>

</div>

<!-- 6o. Campo -->

<h:inputText value="#{capaBean.documentoSeguro.diaMelhorVencimento}" id=*"melhorData"* onblur=*"checkDate(this);"*

onkeypress="return maskData(event,this), desabilitarEnter(event)"

converter=*"dateConverter"* />

</h:panelGroup>

#### Máscaras

Incluir no xhtml a utilização do arquivo app.js.

Data : java.util.Date - class="*date*" converter=*"dateConverter"*

Moeda: java.math.BigDecimal - class="*moeda*" converter=*"moedaConverter"*

Apenas Letras: java.lang.String - class="*apenasLetras*"

Apenas Números: class="numerico"

CPF: java.lang.String - class="*cpf*" converter=*"cpfConverter"*

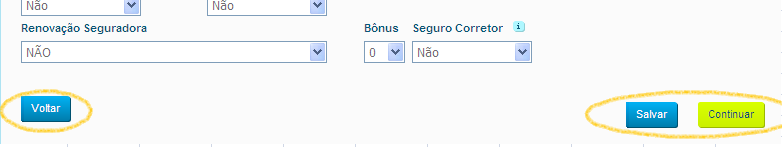
CNPJ: java.lang.String - class="*cnpj*" converter=*"cnpjConverter"*

Telefone: java.lang.String - class="*telefone*" converter=*"telefoneConverter"*

Na inicialização do formulário deve ser executado o javascript: FormUtils.aplicarMascaras('#form\_id');

#### Botões

* Navegação



*btCor1* (Azul, pode mudar dependendo do tema parametrizado pelo usuário):

<a4j:outputPanel ajaxRendered=*"true"* layout=*"block"* styleClass=*"btCor1"* >

<a4j:commandLink id=*"meuBotao"* ignoreDupResponses=*"true"* ajaxSingle=*"true"*

value=*"#{form.meuBotao}"*>

</a4j:commandLink>

</a4j:outputPanel>

*btCor4* (Verde, pode mudar dependendo do tema parametrizado pelo usuário):

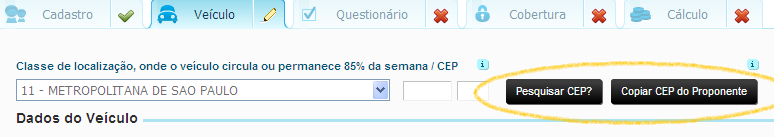
<div class=*"btCor4 col-lado-a-lado"*>

<a href=*"javascript:ExemploController.continuar();"*

id=*"btnsalvarContinuarCapa"* >#{msgBundle.btnSalvarContinuar}</a>

</div>

* Ação em formulário (*btCor3*)



<div class=*"btCor3 col-lado-a-lado"*>

<a4j:commandLink title=*"#{form.meuBotao.title}"* id=*"salvarAddItem"*

value=*"#{form.meuBotao}"*

action=*"#{controller.minhaAction}"* />

</div>

#### Binding com o Controller

As páginas .xhtml não devem possuir qualquer regra de controller, apenas binding para propriedades que representam essa regra.

Exemplo errado:

<h:selectOneMenu value=*"#{controlePagamento.nacionalidade}"*

id=*"nacionalidade"* class=*"select-medio"* **disabled=*"#{controlePropostaCapa.documentoIdentificacao.cadastroTipoDocumentoIdentificacaoPessoa.tipoDocumentoIdentificacao eq 11 ||controlePropostaCapa.documentoIdentificacao.cadastroTipoDocumentoIdentificacaoPessoa.tipoDocumentoIdentificacao eq 12 }"*>**

<f:selectItems value=*"#{controlePagamento.listaNacionalidade}"*/>

</h:selectOneMenu>

O erro está na propriedade “disabled”, acumulando regras de controle na página.

Exemplo corrigido:

<h:selectOneMenu value=*"#{controlePagamento.nacionalidade}"*

id=*"nacionalidade"* class=*"select-medio"* **disabled=*"#{controlePagamento.nacionalidadeDesabilitada}"*>**

<f:selectItems value=*"#{controlePagamento.listaNacionalidade}"*/>

</h:selectOneMenu>

#### Javascript

As páginas não devem conter regras de controlle javascript, apenas invocar uma única função com assinatura bem definida responsável pelo controle do fluxo de execução.

Exemplo errado:

<h:selectOneMenu value=*"#{controlePagamento.nacionalidade}"*

id=*"nacionalidade"* class=*"select-medio"* disabled=*"#{controlePagamento.nacionalidadeDesabilitada}"*

**onchange=*"changeNaturalidade(); setarFlagRecalculoGlobal(true);"*>**

<f:selectItems value=*"#{controlePagamento.listaNacionalidade}"*/>

</h:selectOneMenu>

O erro está no evento onchange, concentrando regras de controlle.

Exemplo corrigido:

<h:selectOneMenu value=*"#{controlePagamento.nacionalidade}"*

id=*"nacionalidade"* class=*"select-medio"* disabled=*"#{controlePagamento.nacionalidadeDesabilitada}"*

**onchange=*"PagamentoController.nacionalidadeAlterada();"*>**

<f:selectItems value=*"#{controlePagamento.listaNacionalidade}"*/>

</h:selectOneMenu>

* “Número Mágicos”

Assim como no servidor, no Javascript todas as constantes devem possuir nomes bem definidos e concentrados em uma classe de acordo com seu significado de negócio.

Exemplo errado:

**function** limparRamal(nomeTipoContato, nomeRamal) {

**var** tipoContato = document.getElementById(nomeTipoContato);

**var** ramal = document.getElementById(nomeRamal);

**if**(tipoContato && (tipoContato.value == 3 || tipoContato.value == 11)) {

**if**(ramal){

ramal.style.display='none';

}

} **else** {

**if**(ramal){

ramal.style.display='block';

}

}

}// O que são os valores 3 e 11??

Exemplo corrigido:

**var** TipoContato = {RESIDENCIAL : 3, CELULAR: 11};

**var** MeuFormularioController = {

limparRamal : **function** (idTipoContato, idRamal) {

**var** tipoContato = document.getElementById(idTipoContato);

**var** ramal = document.getElementById(idRamal);

**if**(ramal){

**if**(tipoContato &&

(tipoContato.value == TipoContato.RESIDENCIAL

|| tipoContato.value == TipoContato.CELULAR)) {

ramal.style.display='none';

} **else** {

ramal.style.display='block';

}

}

}

};

#### Validação

##### Javascript

A aplicação utiliza como padrão para validação no cliente o jQuery Validation Plugin 1.8.1.

Incluir no xhtml a utilização do arquivo app.js.

A validação deve ser encapsulada em objeto javascript correspondente ao formulário.

Exemplo:

**var** AbaAController = {

ID: '#form\_id',

init : **function** () {

FormUtils.aplicarMascaras(AbaAController.ID);

},

isValid : **function** () {

jQuery(AbaAController.ID).validate({

rules:{

'form\_id:nomeCliente':{

required: **true**,

minlength: 2

}

},

messages:{

'form\_id:nomeCliente':{

required: "Informe o nome do Cliente",

minlength: "Informe um nome com no m\u00EDnimo {0} caracteres"

}

},

errorLabelContainer: "#messageBox

});

**if**(jQuery(AbaAController.ID).validate().form()){

**return** **true**;

}**else**{

App.rolarParaTopo();

**return** **false**;

}

}

};

##### JSF

Utilizar o método AbstractFormBean.setErrorMessage(UIComponent componente, String mensagem).

#### Mensagens

As mensagens podem ser exibidas no carregamento de uma página utilizando o atributo “rendered” ou através de javascript no retorno de uma “jsFunction”.

Para tratamento no retorno de uma “jsFunction”, o sistema utiliza como padrão um wrapper para mensagens: com.porto.portoprint.automovel.controller.Mensagem.java. Cada mensagem é tratada de acordo com atributo “tipoMensagem” que é do tipo com.porto.portoprint.automovel.domain.enums.TipoMensagem.

#### Utilização de Ajax

O sistema utiliza o richfaces para requisições via Ajax.

* “reRender” : utilizado para atualizar determinada área da tela, ou seja, um botão que possui esse método informa ao servidor que após sua conclusão a área informada deve ser atualizada com o estado do servidor. Seu uso deve ser sempre bem ponderado pois pode aumentar consideravelmente o número de requisições executadas, bem como o tráfego de rede utilizado. Caso necessário, sempre deve vir acompanhado da propriedade “*limitToList=’true’”.* Uma alternativa é a utilização de uma “jsFunction” e executar seu tratamento via javascript.
* “a4j:support”: sempre que utilizado deve possuir os atributos ajaxSingle="true" e ignoreDupResponses="true". Já os atributos data e process devem ser utlizados conforme a necessidade.
* “a4j:jsFunction”: utilizado para fazer uma requisição ao servidor via javascript.

OBS.: para tratar um retorno específico, ao invés de utilizar o método reRender, pode ser utilizada a propriedade “data” para armazenar a resposta da execução e, no método oncomplete, referenciar o retorno para tratamento.

Exemplo:

<a4j:jsFunction id=*"minhaAcao"*

name=*"MeuFormularioJS.meuMetodoJavascript"*

action=*"#{meuController.minhaAcaoNoServer}"*

data=*"#{meuController.respostaDaAcaoExecutada}"*

oncomplete=*"MeuFormularioJS.tratarRespostaViaJS(data);"* >

</a4j:jsFunction>

Detalhes:

“name”: método que executará o Ajax, no caso sua chamada será M*euFormularioJS.meuMetodoJavascript();*

“action”: método que será chamado no lado servidor

“data”: atributo no servidor que armazenará a resposta do método definido no atributo “action”

“oncomplete”: método javascript que tratará o retorno, note que ele recebe o atributo data como parâmetro. O richfaces automaticamente criará um objeto javascript igual o definido no atributo “data” para que seja possível o tratamento no lado cliente.

#### Fila Global para Ajax

No ppweb é utilizado o parâmetro **org.richfaces.queue.global.enabled** que é incluído no web.xml. Conforme documentação do RichFaces este parâmetro é utilizado para enfileirar as requisições ajax da aplicação. A documentação do RichFaces informa que desta forma as requisições ajax tornam-se síncronas.

Fazemos isso, pois não utilizamos navegação do JSF. Nossa navegação é toda feita por “reRender”, sendo assim necessitamos que as requisições sejam síncronas e dependentes (uma requisição depende da finalização da requisição anterior).

Este parâmetro não deve ser alterado sem um estudo detalhado e devidos testes dos impactos no comportamento do PPWEB.

#### Utilização de a4j:pool

O componente a4j:pool é utilizado quando é necessário que sejam feitas requisições assíncronas de “tempos em tempos” ao servidor. Cada requisição que é feita é utilizada uma thread do webcontainer do servidor de aplicação. Estas threads são as mesmas utilizadas para atender as requisições de salvar, continuar, calcular, etc. Podendo então causar um impacto no ambiente, aumentando o consumo de threads e causando lentidão.

**Não recomendamos a utilização deste componente. Caso seja necessário em algum projeto, devem antes serem feitos testes de stress, performance e monitoramento do ambiente juntamente com a equipe de sistemas.**

#### Utilização de a4j:queue

O Richfaces disponibiliza algumas funcionalidades para trabalhar com filas diferentes na aplicação. Permitindo que o processamento de uma ação seja realizado de forma síncrona ou assíncrona. Criando múltiplas filas é possível que fazer controles do tipo: Requisição A, seja síncrona com Requisição B, e esteja executando paralelamente com a Requisição C.

Quando se trabalha desta forma para a Requisição A utilizará uma thread (a Thread 1, por exemplo) e a requisição C outra (a Thread 2, por exemplo). A requisição B pode ser que utilize a mesma Thread 1, visto que é síncrona.

Neste cenário então, uma única ação do usuário utiliza 2 threads no servidor. Para efetuar o controle de múltiplas filas, normalmente são utilizadas funções javascript. Porém, na concorrência de requisições nem sempre é possível visto que o javascript é uma requisição síncrona.

**O uso indiscriminado desta funcionalidade pode degradar o ambiente quando houver grande concorrência de usuários (horários de pico). Por isso também não recomendamos a utilização destas filas no sistema. Caso seja necessário em algum projeto, devem antes serem feitos testes de stress, performance e monitoramento do ambiente juntamente com a equipe de sistemas.**

### Navegação

#### Menus

O menu principal do sistema fica concentrado no arquivo header.xhtml, que é importado pela página do template.xhtml e portanto é apresentada em todas as páginas do sistema.

Cada item do menu deve sempre fazer chamada ao bean NavegacaoController.java e a partir dele fazer as chamadas de inicialização necessárias a cada tela.

#### Abas

As abas devem ser desacopladas, ou seja, cada implementação não deve possuir nenhuma referência/dependência para outras abas.

A implementação deve possuir um controller “pai” responsável pela navegação e comunicação entre elas tanto no JSF quanto no Javascript.

Eventos que gerem impactos em outras abas devem seguir o padrão Event Model, ou seja, a aba origem deve possuir uma lista de listeners e ao gerar o evento, o ouvinte (no caso o “controller pai”), delegar a ação a ser tomada.

O uso do “reRender” deve ser sempre que possível evitado, o retorno pode ser tratado de acordo com o resultado de cada ação via javascript. Caso necessário, sempre deve vir acompanhado da propriedade “*limitToList=’true’*”.

## Camada de Serviço

**O projeto portoprint-automovel-domain**

Responsável pelas interfaces de serviço e os objetos comuns a todas as camadas do sistema (DTOs).

Os serviços deveram ser criados por entidade ou por fluxo de negócio.

**O projeto portoprint-automovel-service**

Responsável pela implementação das interfaces do projeto domain e camada de persistência.

**O projeto portoprint-orm**

Responsável pelo mapeamento objeto relacional da base do PortoprintWeb. É compartilhado por todos os projetos.

### Criando um serviço

1- Criar a interface no projeto domain seguindo o padrão de pacotes:

"com.porto.portoprint.automovel.application.service".

Exemplo:

**package** com.porto.portoprint.automovel.service;

**public** **interface** CapaOrcamentoService {

DadosCapaOrcamento salvarDadosCapaOrcamento(**final** DadosCapaOrcamento dadosCapaOrcamento);

}

Podemos ver na assinatura do método acima que o objeto passado como parâmetro DadosCapaOrcamento é um Wrapper para os dados que compõe determinada regra de negócio.

Veja abaixo o exemplo de assinatura incorreta.

DadosCapaOrcamento salvarDadosCapaOrcamento(**final** DocumentoSeguro documentoSeguro);

O documentoSeguro é muito genérico, possui tudo o que é necessário para a execução da regra de negócio porém é preciso navegar em sua estrutura adicionando lógica e passando dados desnecessários ao fluxo de execução do negócio.

UM MÉTODO DEVE RECEBER APENAS E UNICAMENTE O QUE DE FATO É NECESSÁRIO PARA SUA EXECUÇÃO.

2 - Criar a implementação no projeto service seguindo o padrão de pacotes: "com.porto.portoprint.automovel.service".

Exemplo:

**package** com.porto.portoprint.automovel.service;

@ServiceLayer("capaOrcamentoService")

**public** **class** CapaOrcamentoServiceProvider **implements** CapaOrcamentoService {

**private** **final** EntityStateManager entityStateManager;

@Autowired

**public** CapaOrcamentoServiceProvider(**final** EntityStateManager entityStateManager) {

**this**.entityStateManager = entityStateManager;

}

@Override

**public** DadosCapaOrcamento salvarDadosCapaOrcamento(**final** DadosCapaOrcamento dadosCapaOrcamento) {

Assert.*notNull*(dadosCapaOrcamento, "Dados capa nulo");

...

entityStateManager.markToUpdate(dadosCapaOrcamento.getDocumentoSeguro(););

entityStateManager.markToUpdate(dadosCapaOrcamento.getCadastroPessoa(););

...

**return** dadosCapaOrcamento;

}

}

Os serviços devem sempre ser anotados como @ServiceLayer.

Todas as regras de negócio do sistema devem obrigatoriamente ser implementadas nessa camada.

No código acima nota-se que o objeto documento seguro foi marcado para ser atualizado. Quem controla realiza as operações com o banco de dados é a Unidade de Trabalho de acordo com o tipo de marcação.

### Persistência

Deve ser utilizado um único DataManager por ORM.

Um serviço deve orquestrar quantos DataManagers forem necessários de acordo com determinada regra de negócio, portanto, um DataManager não deve possuir dependência com nenhum outro DataManager ou executar operações com mais de um ORM.

***TODOS*** os DataManagers devem executar apenas operações de SELECT. As demais operações devem ser executadas nos serviços utilizando a Unidade de Trabalho.

***NENHUM*** DataManager deve possuir regra de negócio, caso necessário, alterar a projeção para apenas os campos desejados e passar os parâmetros da query na assinatura dos métodos.

Exemplo incorreto:

**public** **class** OperacaoEmpresaDataManager **extends**

AbstractDataManager<OperacaoEmpresa, OperacaoEmpresaPK> **implements** OperacaoEmpresaDao {

...

**private** **void** gravarOperacaoEmpresa(**final** OperacaoEmpresa operacaoEmpresa) {

**if**(!operacaoEmpresa.getComp\_id().getCodigoEmpresaOperacao().equals(

PoeEnum.*EMPRESA\_POE\_DEFAULT*.getCodigo())){

operacaoEmpresa.setNomeEmpresaOperacao(**null**);

}

merge(operacaoEmpresa);

}

...

}

No exemplo acima observa-se que dentro do escopo do método ***gravarOperacaoEmpresa(...)*** está verificando determinado atributo para efetuar então uma alteração na ORM que será persistida. Isso caracteriza uma regra de negócio. Sendo assim, esta foi incluída em local incorreto. Esta alteração da ORM deve ser feita na camada de serviço antes que a ORM seja passada como parâmetro.

***TODOS*** os DataManagers devem implementar uma interface como padrão de nomenclatura o sufixo ‘Dao’ e com a interface no pacote “com.porto.portoprint.automovel.dao” no projeto ***portoprint-automovel-domain***.

#### Criando um DAO

Criar uma classe que extenda a classe AbstractDataManager<?, ?> e implemente uma interface Dao. Exemplo:

**package** com.porto.portoprint.automovel.dao;

**public** **class** CadastroPessoaDataManager **extends** AbstractDataManager<CadastroPessoa, Integer> **implements** CadastroPessoaDao {

...

**public** CadastroPessoa obterCadastroPessoaById(**final** Integer codigoPessoa) {

**final** Criteria criteria = getManager().createCriteria(getEntityClass());

criteria.createAlias("documentoIdentificacaos",

"documentoIdentificacaos", Criteria.*LEFT\_JOIN*);

criteria.createAlias("emailClientes", "emailClientes",Criteria.*LEFT\_JOIN*);

criteria.createAlias("contatoClientes", "contatoClientes", Criteria.*LEFT\_JOIN*);

criteria.createAlias("enderecoClientes", "enderecoClientes", Criteria.*LEFT\_JOIN*);

criteria.createAlias("cadastroVinculoPessoas","cadastroVinculoPessoas", Criteria.*LEFT\_JOIN*);

criteria.add(Restrictions.*eq*("codigoPessoa", codigoPessoa));

**return** (CadastroPessoa) criteria.uniqueResult();

}

...

}

Como pode ser observado esta classe deve possuir o sulfixo ***DataManager***  e ser incluída no projeto ***portoprint-automovel-service*** e a interface que ela implementa deve possuir o sulfixo ***Dao*** e ser incluída no projeto ***portoprint-automovel-domain***.

#### Efetuando cópia de entidades

Muitas vezes é preciso efetuar a cópia de uma entidade recuperada do banco, alterar os seus dados para persistir novas informações.Entretanto, devido aos inúmeros relacionamentos que uma entidade pode possuir, é preciso fazer uma cópia profunda (*deep copy*) da entidade. Ao efetuar a cópia profunda, evitamos que duas ou mais referências existam para um mesmo objeto, o que poderia causar erros de persistência e até de consistência dos dados durante a execução da tarefa solicitada.

Uma das opções utilizadas para esse tipo de cópia é a serialização e de-serialização de um objeto, método que irá gerar um novo objeto sem nenhuma referência com o objeto original.

Para facilitar essa tarefa, utilizamos o método *clone(Object obj)* da classe *SerializationUtils* dopacote *org.apache.commons.lang,* dependência já existente no projeto.

Exemplo:

**public** CadastroPessoa clonarCadastroPessoa( **final** CadastroPessoa cadastroPessoa) {

**final** CadastroPessoa destCadastroPessoa =

(CadastroPessoa) SerializationUtils.*clone*(cadastroPessoa);

**return** destCadastroPessoa;

}

Por se tratar de uma serialização, o *cast* do objeto copiado é necessário.

Esse método de cópia deve ser utilizado em qualquer situação onde se deseja uma cópia de objetos que não compartilhem referências entre si.

#### Criando buscas em entidades (Hibernate)

Para efetuar a busca em entidades do hibernate devemos estar atentos aos tipos de coleções que estão relacionadas com aquela entidade. Por padrão as entidades que tem relacionamento **one-to-one** com uma entidade pai tem seu *fetching* configurada para *EAGER*. Com isso as entidades filhas tem seus joins incluídos na query da entidade pai. Segue abaixo um exemplo:

* Pesquisa criada no DAO:

**final** Criteria criteria1 = **super**.getManager().createCriteria(DocumentoSeguro.**class**);

criteria1.list();

* Select Gerado:

SELECT \*

FROM APOMSEGDOC this\_,

GSCMAZLSEGDOC documentos2\_,

APOMITASEGDOC documentos3\_,

APOMCRTRNVPGTSUG sugestaopa4\_,

GSCMCRDCRTSLC solicitaca5\_,

APOREDSAPL apoliceend6\_,

APOREDSAPLORIVCL veiculoori7\_,

APOKBCOCNLOPE operacaoca8\_,

APOMIDVOFR ofertaindi9\_

WHERE this\_.SEGDOCCOD=documentos2\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =documentos3\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =sugestaopa4\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =solicitaca5\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =apoliceend6\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =veiculoori7\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =operacaoca8\_.SEGDOCCOD(+)

AND this\_.SEGDOCCOD =ofertaindi9\_.ORISGRDCTCOD(+)

Notamos que na pesquisa criada foram incluídos vários joins desnecessários. Porém, estes consomem o banco de dados e deixam complexa, uma query que era simples. Para que isso não ocorra pode-se setar o *fetchMode* de uma determinada entidade, que esteja como one-to-one, para LAZY utilizando o comando ***criteria.setFetchMode(...)*** da seguinte forma ***criteria.setFetchMode("<nome do atributo>", FetchMode.LAZY);*** Para facilitar este mapeamento das entidades foram criados na classe com.porto.portoprint.automovel.dao.AbstractDataManager<T, ID> métodos responsável por setar o fetch mode para Lazy nas entidades pais. Um exemplo de utilização do mesmo encontra-se abaixo, alterando a Criteria acima:

* Pesquisa alterada no DAO:

**final** Criteria criteria = **super**.getManager().createCriteria(DocumentoSeguro.**class**);

**setFetchModeLazyDocumentoSeguro(criteria);**

criteria.list();

* Select gerado

SELECT …

FROM APOMSEGDOC this\_

Segue o detalhe do método **setFetchModeLazyDocumentoSeguro(...);**

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "documentoSeguroAzul"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "documentoSeguroIsar"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "sugestaoPagamentoRenovacaoCartaoCredito"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "solicitacaoCartaoCredito"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "apoliceEndosso"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "veiculoOriginalApoliceEndosso"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "operacaoCanalBanco"), FetchMode.~~LAZY~~);

criteria.setFetchMode(concatAlias(aliasDocumentoSeguro, "ofertaIndividual"), FetchMode.~~LAZY~~);

Neste método é alterado o *fetching* das entidades filhas de documentoSeguro.

**Se houver alguma inclusão de um atributo on-to-one em alguma entidade deve ser também alterado este método para incluir o atributo em questão.**

### O que é a Unidade de Trabalho

A unidade de trabalho foi criada para controlar todas as operações de persistência na base de dados, minimizando assim problemas de lock e dificuldade de rastreamento de erros por operação.

Para a Unidade de Trabalho ser injetada, a classe deve possuir a anotação @ServiceLayer e deve possuir construtor recebendo a dependência via anotação Autowired fornecida pelo Spring.

#### Utilizando a Unidade de Trabalho

Abaixo temos um exemplo de uso a Unidade de Trabalho, onde os objetos são marcados para serem atualizados, persistidos ou deletados.

...

@Override

**public** DadosCapaOrcamento salvarDadosCapaOrcamento(**final** DadosCapaOrcamento dadosCapaOrcamento) {

...

**if** (cadastroPessoa.getCodigoPessoa() == **null**) {

entityStateManager.markToInsert(cadastroPessoa);

documentoIdentificacao.setCadastroPessoa(cadastroPessoa);

documentoIdentificacao.getComp\_id().setCodigoPessoa

(cadastroPessoa.getCodigoPessoa());

entityStateManager.markToInsert(documentoIdentificacao, cadastroPessoa);

entityStateManager.markToInsert(enderecoCliente, cadastroPessoa);

} **else** {

entityStateManager.markToUpdate(cadastroPessoa);

entityStateManager.markToUpdate(documentoIdentificacao);

}

documentoSeguro.setCadastroPessoa(cadastroPessoa);

entityStateManager.markToUpdate(documentoSeguro);

...

}

...

No exemplo acima nenhum DAO é chamado, quem irá cuidar da persistência é a Unidade de Trabalho.

Há vários métodos que podem ser utilizados, dentre eles temos os que recebem como parâmetros listas ou objetos que possuem relação entre si para persistência em cascata. Cabe ao desenvolvedor **ler** o javadoc e utilizar o método adequado para o serviço em desenvolvimento.

#### Rastreando execução da unidade de trabalho

Para logar a execução da unidade de trabalho é necessário configurar o log4j com threshold ‘TRACE’ no pacote ‘com.porto.portoprint.automovel.persistence’.

No Jboss utilizado em desenvolvimento a configuração fica no arquivo conf/jboss-log4j.xml do profile porto. Exemplo:

<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">

<errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>

<param name="Target" value="System.out"/>

<param name="Threshold" value="ALL"/>

<layout class="org.apache.log4j.PatternLayout">

<!-- The default pattern: Date Priority [Category] Message\n -->

<param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c{1}] (%t:%x) %m%n"/>

</layout>

</appender>

<category name="com.porto.portoprint.automovel.persistence">

<priority value="TRACE"/>

</category>

### Padrão de mensagens

Para tratamentos de mensagens geradas pelos serviços, o sistema deve possuir um enum concentrador de mensagens por fluxo de negócio, um wrapper de mensagens de negócio por tipo e um wrapper de mensagens na camada web.

#### Tipos de mensagens

Os tipos são definidos pelo enum com.porto.portoprint.automovel.domain.enums.TipoMensagem.

**public** **enum** TipoMensagem {

*OK*(1), *ALERTA*(2), *ERRO*(3, **true**), *ERRO\_VALIDACAO*(4, **true**);

**private** **int** value;

**private** **boolean** erroImpeditivo = **false**;

**private** TipoMensagem(**final** **int** value) {

**this**.value = value;

}

**private** TipoMensagem(**final** **int** value, **final** **boolean** erroImpeditivo) {

**this**.value = value;

**this**.erroImpeditivo = erroImpeditivo;

}

**public** **int** getValue() {

**return** value;

}

**public** **boolean** isErroImpeditivo(){

**return** **this**.erroImpeditivo;

}

}

Cada tipo possui um código e um atributo booleano para informar se é impeditivo para continuidade de determinada regra de negócio.

#### Definição das mensagens

**package** com.porto.portoprint.automovel.domain.enums.exemplo;

**public** **enum** MensagemExemploEnum {

*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ALERTA*,

*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ERRO*,

*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ERRO\_VALIDACAO*,

*EXEMPLO\_ABA\_A\_DATA\_VIGENCIA\_ALTERADA*,

*EXEMPLO\_ABA\_A\_EXIBE\_MENSAGEM\_QUE\_VAI\_PARA\_QUESTIONARIO*;

}

#### Wrapper para mensagens por tipo

**public** **class** MensagemExemploA {

**private** List<MensagemExemploEnum> mensagens = **new** LinkedList<MensagemExemploEnum>();

**private** **boolean** erroImpeditivo = **false**;

**private** TipoMensagem tipo;

**public** MensagemExemploA(**final** TipoMensagem tipoMensagem) {

**this**.tipo = tipoMensagem;

**this**.erroImpeditivo = tipo.isErroImpeditivo();

}

**public** **void** addMensagem(**final** MensagemExemploEnum mensagemExemploA) {

**this**.mensagens.add(mensagemExemploA);

}

**public** **void** clearMensagens() {

**this**.mensagens.clear();

}

**public** List<MensagemExemploEnum> getMensagens() {

**return** mensagens;

}

**public** **boolean** isErroImpeditivo(){

**return** **this**.erroImpeditivo;

}

**public** TipoMensagem getTipo() {

**return** tipo;

}

}

#### Exemplo de implementação no serviço

**...**

MensagemExemploA mensagem = **new** MensagemExemploA(TipoMensagem.*ALERTA*);

mensagem.addMensagem(MensagemExemploEnum.*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ALERTA*);

mensagem.addMensagem(MensagemExemploEnum.*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ALERTA\_2*);

dadosCapaOrcamento.addMensagem(mensagem);

…

mensagem = **new** MensagemExemploA(TipoMensagem.*ERRO\_VALIDACAO*);

mensagem.addMensagem(MensagemExemploEnum.*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ERRO\_VALIDACAO*);

dadosCapaOrcamento.addMensagem(mensagem);

…

mensagem = **new** MensagemExemploA(TipoMensagem.*ERRO*);

mensagem.addMensagem(MensagemExemploEnum.*EXEMPLO\_ABA\_A\_MINHA\_MENSAGEM\_ERRO*);

dadosCapaOrcamento.addMensagem(mensagem);

#### Wrapper para mensagens na Web

Implementado na classe com.porto.portoprint.automovel.controller.Mensagem.

### Arquivos de configuração

Os arquivos de configuração ficam na pasta resources/properties.

Cada arquivo de configuração deve possuir uma única classe para manipulá-la.

Como padrão é utilizado um enum com cada entrada do arquivo e um método value() que retorna o valor informado no arquivo. Segue exemplo para o arquivo parametrosgerais.properties:

**package** com.porto.portoprint.automovel.enums;

**import** com.porto.infra.util.ConfigProperties;

**public** **enum** ParametrosSistema {

/\*\* The Constant APARENCIA\_PADRAO. \*/

APARENCIA\_PADRAO("APARENCIA\_PADRAO"),

/\*\* Diretorio \*/

DIRETORIO("DIRETORIO");

/\*\*

\* Define caminho do arquivo de properties

\*/

**private** **static** **final** String PROPERTIES\_CONF = "properties/parametrosgerais";

/\*\*

\* Arquivo properties

\*/

**private** **static** **final** ConfigProperties config = **new** ConfigProperties(PROPERTIES\_CONF);

/\*\*

\* Chave da configuração do sistema

\*/

**private** **final** String key;

**private** ParametrosSistema(**final** String key) {

**this**.key = key;

}

**public** String value() {

**return** config.readString(key);

}

**public** String key() {

**return** key;

}

}

# Implementação de exemplo

A branch de desenvolvimento possui a viewID exemplo.xhtml que tem por objetivo organizar implementações de referência para cenários comuns no sistema.

## Implementando uma aba

As páginas xhtmls seguem a seguinte:

1. nav\_abas\_exemplo.xhtml:

Concentra todas as abas decorando o template */template/nav\_abas.xhtml*.

1. exemplo\_aba\_a.xhtml:

Formulário presente na aba A. Utiliza o template */template/aba.xhtml*.

1. exemplo\_aba\_b.xhtml:

Formulário presente na aba B. Utiliza o template */template/aba.xhtml*.

O javascript de o exemplo foi implementado em apenas um arquivo chamado exemplo.js e nele estão as implementações dos controles das abas A e B, bem como suas funções de navegação, exibição de modais e validação.

As classes controller são (JSF e Javascript):

1. ExemploController.

Responsável pelo controle do componente de abas (navegação, ações e comunicação entre elas).

1. AbaExemploAController:

Responsável pelo comportamento do formulário da Aba A.

1. AbaExemploBController:

Responsável pelo comportamento do formulário da Aba B.

### Navegação

A mudança de aba é realizada em conjunto entre o javascript e o Java. De acordo com a ação executada e com o estado da aplicação é chamado o javascript ou a navegação pelos Managed Beans.

Na implementação de exemplo as classes são: com.porto.portoprint.automovel.controller.exemplo.ExemploController e exemplo.js.ExemploController.

## Eventos

De acordo com regras de negócio, telas e abas podem afetar outras telas e abas do sistema. Seguindo o padrão Event Model a implementação de exemplo possui a seguinte estrutura:

* **Interface Listener:** Define os eventos que podem ocorrer em uma tela;
* **Source (Origem):** Origem do evento, responsável por sinalizar a todos os listeners a sua ocorrência.
* **Listener(Ouvinte):** Classe que implementa a *Interface Listener*.

### Interface Listener

A aba ‘A’ possui os eventos para sinalizar quando foi salva e quando o valor do campo tipo de documento foi alterado.

**public** **interface** AbaExemploAControllerListener **extends** EventListener {

**public** **void** onSalvar(DadosExemploAOrcamento dadosExemploAOrcamento);

**public** **void** tipoDocumentoAlterado(TipoDocumentoCapaEnum tipoDocumentoCapaEnum);

}

### Source

A classe origem dos eventos, no caso a aba ‘A’, deve possuir uma lista de listeners (ouvintes) e, a cada evento definido na interface, sinalizar sua ocorrência.

#### O Announcer

Classe utilitária para gerenciar (armazena e sinaliza) todos os listeners definidos a partir de uma interface de eventos.

#### Exemplo

**public** **class** AbaExemploAController **extends** AbaController {

**protected** Announcer<AbaExemploAControllerListener> listeners =

Announcer.*to*(AbaExemploAControllerListener.**class**);

@Override

**public** ControllerRespostaOrcamento salvar() {

listeners.announce().onSalvar(dadosCapa);

**return** controllerResposta;

}

**public** **void** addListener(**final** AbaExemploAControllerListener listener){

**this**.listeners.addListener(listener);

}

}

### Listener

O ExemploController.java que é o responsável por todas as abas e é quem deve tratar o evento e executar o devido tratamento.

**public** **class** ExemploController {

@PostConstruct

**public** **void** init() {

inicializarEventos();

}

**private** **void** inicializarEventos(){

tratarEventosAbaExemploA();

}

**private** **void** tratarEventosAbaExemploA(){

**this**.abaExemploAController.addListener(**new** AbaExemploAControllerListener() {

**public** **void** onSalvar(**final** DadosExemploAOrcamento capaOrcamento) {

System.*out*.println("SALVAR ABA");

}

**public** **void** tipoDocumentoAlterado(**final** TipoDocumentoCapaEnum tipoDocumento) {

addReRender(

abaExemploBController

.atualizarExibicaoInformacoesApolice(tipoDocumento));

}

});

}

}

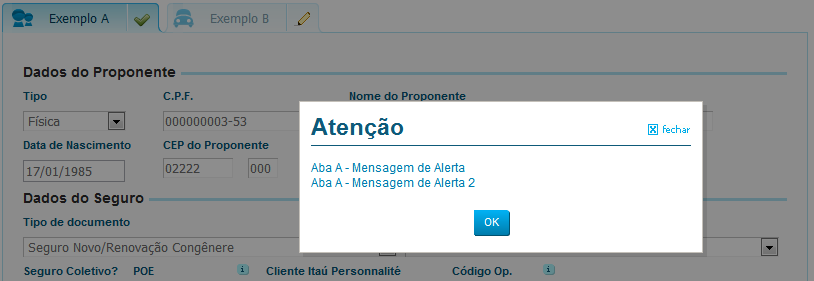
## Cenários de exemplo

A implementação de exemplo contempla diversos cenários comuns no PortoprintWeb, segue abaixo breve descrição de cada um.

#### Mensagens de alerta entre abas

Inicia na aba ‘A’ e é acionado ao salvar quando informado no campo de nome o valor ‘CENARIO UM’.

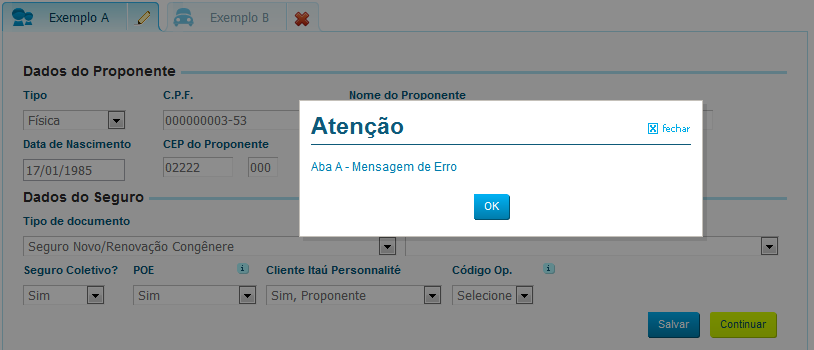
Comportamento esperado: exibição de popup de alerta com duas mensagens, ao clicar em ‘OK’, o sistema continua o fluxo de navegação.



#### Mensagem de erro que impede continuação do fluxo de navegação

Inicia na aba ‘A’ e é acionado ao salvar quando informado no campo de nome o valor ‘CENARIO DOIS’.

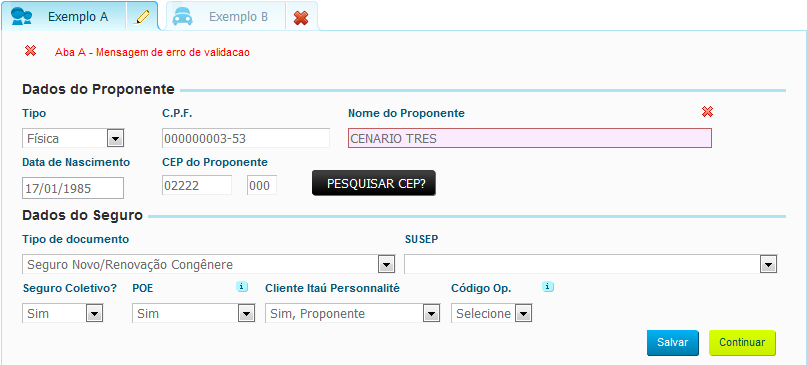
Comportamento esperado: exibição de popup de alerta com mensagem de erro e, ao clicar em ‘OK’, o sistema não permite continuidade no fluxo de navegação.



#### Mensagem de validação que impede continuação do fluxo de navegação

Inicia na aba ‘A’ e é acionado ao salvar quando informado no campo de nome o valor ‘CENARIO TRES’.

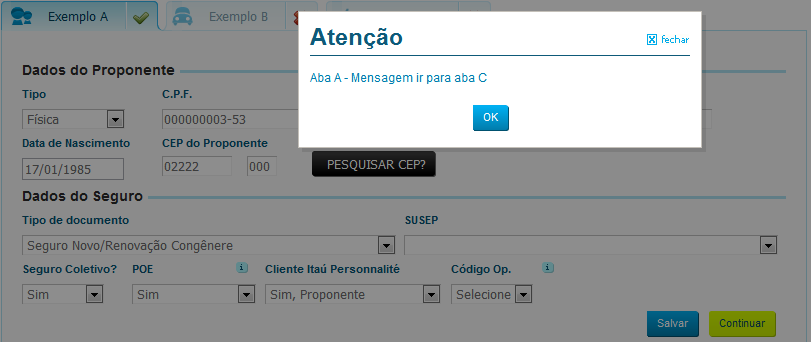
Comportamento esperado: exibição de mensagem de erro de validação retornado pelo serviço de salvar no campo nome do proponente.



#### Mensagem de alerta que muda o fluxo de navegação

Inicia na aba ‘A’ e é acionado ao salvar quando informado no campo de nome o valor ‘CENARIO QUATRO’.

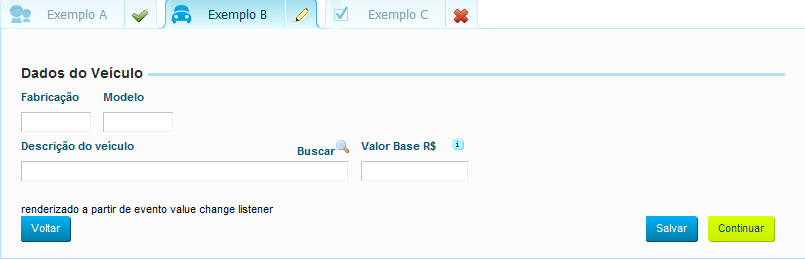
Comportamento esperado: exibição de duas mensagens de alerta e direciona a navegação para a terceira aba.



#### ValueChangeListener + reRender: Em outra aba a partir de campo alterado

A partir da aba ‘A’, é acionado ao alterar o campo “Tipo de documento” para o valor ‘Renovação Porto’.

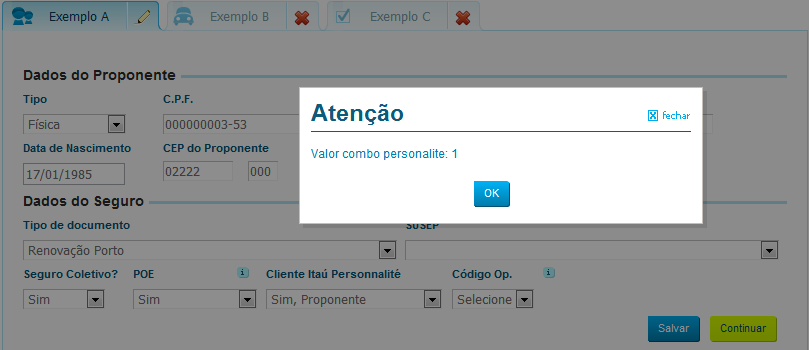
Comportamento esperado: exibição do texto “renderizado a partir de evento value change listener” na aba ‘B’.



#### A4j:support: Tratamento javascript a partir de processamento de um campo via Ajax.

A partir da aba ‘A’, é acionado ao alterar valor do campo ‘Cliente Itaú Personalité’.

Comportamento esperado: exibição de modal com mensagem recebida do servidor.



# Nomenclatura

## 6.1 Nomeação de classes

VOs deverão ter o sufixo **VO** no nome.

Enums deverão ter o sufixo **Enum** no nome

## 6.2 Nomeação de métodos

### 6.2.1 DAO

Remoção: **remover**<ORM>

Busca: **buscar**<ORM>**Por**<Atributos de entrada>

### 6.2.2 Service

Atualização: **atualizar**<ORM>

Inserção: **salvar**<ORM>

# Empacotamento e Importação

## VO

POL deverão estar dentro do pacote dto.pol

Relatório deverão estar dentro do pacote dto.relatorio

Tela:

Para o Projeto portoprint-automovel-domain

As classes deverão estar no pacote com.porto.portoprint.automovel.dto.formulario

Devem importar somente classes do pacote DTO e classes padrão do Java.

## Classes utilitárias

Deverão estar dentro do pacote projeto.service.util

## Serviços

A interface deverá estar no projeto portoprint-automovel-domain

A implementação deverá estar no projeto portoprint-automovel-service

Ambos no pacote com.porto.portoprint.automovel.service

# Chave de Controle de Projeto Piloto

(Parte da documentação que segue foi extraída da própria documentação original do Projeto Gatilho)

Visão Geral

## Visão Geral

Estrutura que permite o cadastro de projeto e sua liberação gradativa.

É possível definir o período de implantação do projeto, após esse período o projeto estará disponível para todos os usuários do PPWeb.

Após definido o período de implantação (estará em fase de liberação gradativa), poderá ser definido para quais sucursais, suseps ou classes de localização (em algumas situações) o projeto estará liberado.

**Exemplo (apenas ilustrativo):**

Projeto Piloto Ativo (Tabela **APOKPILPRJ**):

**Cálculo Isa+r no Porto Print Web**

Sucursal liberada (Tabela **APOKSUCPRJLBR**):

**10 – Sucursal Belém**

Susep liberada (Tabela **APOKSUSPRJLBR**):

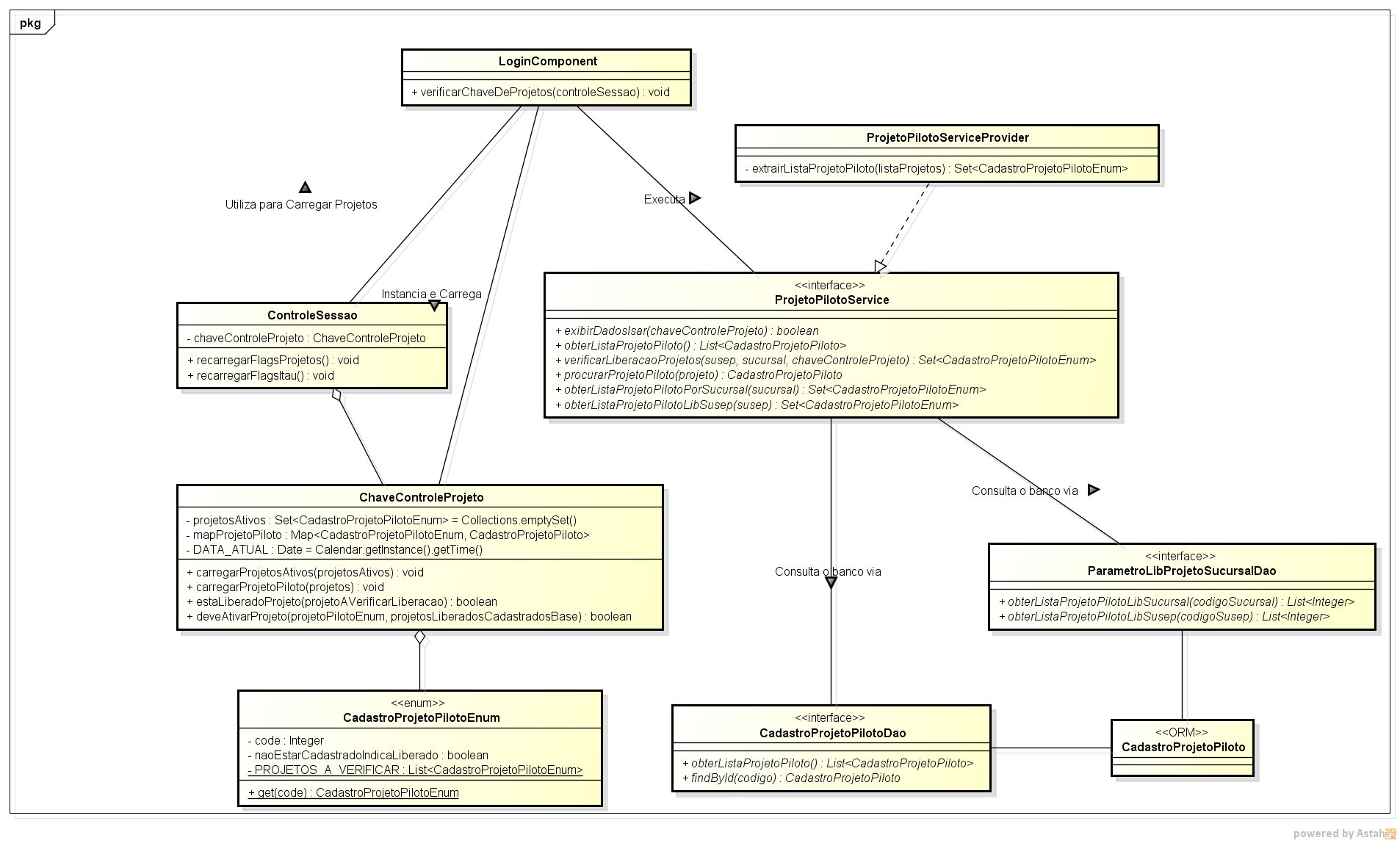
**S5005J**

**Resultado esperado:**

O projeto estará liberado para todas as suseps da sucursal 10 e além destas para a susep S5005J e todas as susepinhas ligadas a ela, não estando disponíveis para as demais.

Ao remover o registro da tabela **APOKPILPRJ** o projeto estará liberado para todas as suseps.

## Diagrama de classes



## Verificação de Projetos Ativos

Os projetos que tem suas chaves de piloto verificadas na aplicação devem ser incluídas no atributo **PROJETOS\_A\_VERIFICAR** do enum **CadastroProjetoPilotoEnum**. Este atributo obtém a referência de uma lista que não deve ser alterada pela aplicação. Ela é iniciada em um bloco estático. Exemplo abaixo:

**public** **static** **final** List<CadastroProjetoPilotoEnum> *PROJETOS\_A\_VERIFICAR*;

**static**{

*PROJETOS\_A\_VERIFICAR* = Arrays.*asList*(

**new** CadastroProjetoPilotoEnum[]{

CadastroProjetoPilotoEnum.*PROJETO\_AUTO\_RE*,

CadastroProjetoPilotoEnum.*PROJETO\_PORTO\_CONECTA*,

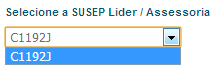
CadastroProjetoPilotoEnum.*PROJETO\_BATENTE*,

...

}

Quando o projeto não for mais pilotado (tiver liberado para todas as suseps) nenhum cadastro na base de dados das tabelas referente ao projeto piloto precisam ser mantido.

Os projetos que são liberados por susep ou por sucursal são carregados no login – no método **LoginComponent.verificarChaveDeProjetos()**, que instancia um objeto de **ChaveControleProjeto** que fará o controle dos projetos na aplicação. Esta instância está disponível no objeto **controleSessao** com referência no atributo **chaveControleProjeto** – ou, na alteração do combo da susep líder, conforme imagem abaixo.



NÃO DEVEM SER CRIADAS FLAGS PARA CONTROLE DE PRJETO (SALVO EM EXCEÇÕES ALINHADAS COM O LÍDER DE SISTEMAS). Nos locais onde é necessário controlar comportamento ou exibição de campos de acordo com a ativação de projeto a instância de **ChaveControleProjeto** deve ser obtida do **controleSessao** do usuário e o método **estaLiberadoProjeto** deve invocado passando como parâmetro o enum do Projeto que deve ser verificado. Como o exemplo abaixo:

controleSessao.getChaveControleProjeto().estaLiberadoProjeto(CadastroProjetoPilotoEnum.*PROJETO\_GRADE\_ACEITACAO*)

Se houver necessidade de utilização em um XHTML este método deve ser encapsulado por um método com retorno **boolean** com padrão de nomenclatura **is<NOME DO PROJETO>liberado()** e a EL deste método incluída no XHTML. Conforme exemplo abaixo:

**public** **boolean** isProjetoGradeAceitacaoLiberado(){

**return** controleSessao.getChaveControleProjeto().estaLiberadoProjeto(

CadastroProjetoPilotoEnum.*PROJETO\_GRADE\_ACEITACAO*);

}

NÃO DEVEM SER INCLUÍDOS MÉTODOS NEM ATRIBUTOS NA CLASSE **ChaveControleProjeto** SEM AUTORIZAÇÃO PRÉVIA DE UM LÍDER DE SISTEMA.

# Execuções eventuais e atualitabs.

## Abertura do chamado

**O formulário para ser anexado no chamado** está disponível na hora de abrir o chamado. O mesmo é aberto (para ser salvo posteriormente e anexado), a partir do botão do SCUA, como mostrado abaixo:

### Regras e horários para abertura

#### Atualitab

O atualitab é definido por uma **execução que não passe de um minuto**, ou seja, que seus comandos não alterem muitas linhas, ele pode ser aberto a qualquer momento e entra na fila de execução pela equipe de produção.

**Link para abrir o chamado de atualitab** com os parâmetros já preenchidos:

http://helpdesk/smartcontact/jsp/cad\_chamado.jsp?cod\_empresa=1&nomeContato=&cod\_abertura\_filtro=null&cod\_solicitacao=null&cod\_associacao=5009;5019;5011;3087;&buscaTipoos=S&ind\_reincidencia=N&txt\_titulo=ATUALITAB&txt\_descricao=Senhores,<br/><br/>favor executar o atualitab descrito no anexo. <br/><br/>Obrigado.

#### Execução eventual

A execução eventual, tipicamente é um batch ou instruções SQL que alterem um grande número de registros, a mesma só pode ser executada em duas janelas, das 12H às 14H ou depois das 18H. **Para a execução da mesma, independentemente da janela, o chamado deve ser aberto no mesmo dia e até às 11H.**

Se passar das 11H, e for necessária a execução de um eventual, o chamado pode ser aberto, porém é obrigatório anexar o e-mail de aceite da coordenação.

**Link para abrir o chamado de execução eventual** com os parâmetros já preenchidos:

http://helpdesk/smartcontact/jsp/cad\_chamado.jsp?cod\_empresa=1&nomeContato=&cod\_abertura\_filtro=null&cod\_solicitacao=null&cod\_associacao=5009;5010;5011;3087;&buscaTipoos=S&ind\_reincidencia=N&txt\_titulo= Job Eventual&txt\_descricao=Senhores,<br/><br/>favor executar o eventual descrito no anexo. <br/><br/>Obrigado.

## Projeto para execução de atualitabs

O projeto do programa para execução eventual, configurado pelo Maven, está no pacote portoprint-batch-eventual, tem como classe main a: com.porto.portoprint.automovel.batch.launcher.ExecucaoEventualJobLauncher.

### Binário (JAR)

O jar sempre será o mesmo, **publicando-se para a U19 na pasta ‘/sheeve’, apenas o arquivo com as queries (.sql) e o script shell de execução (.sh).**

O nome do jar gerado com todas as libs internas, configurado pelo maven (pom.xml) é o **danubio-atualitab.jar**, o mesmo já está publicado na U19 e somente será necessário republicar em futuras correções. Verificar antes de abrir o chamado, se o jar ainda está disponível na pasta, pois o pessoal de produção efetua limpeza da mesma eventualmente.

### Arquivo de comandos (.sql)

**O arquivo de queries deve ter a extensão ‘.sql’,** podendo ter comentários (como '--', '#' ou '//') e **deve ter obrigatoriamente a finalização do comando com ';'. Pois assim é a única forma que o batch sabe onde termina o comando atual.**

**Exemplo de arquivo SQL:**

-- ATUALIZA TABELA ITEM AUTOMOVEL PARA REMOVER ATIVIDADE PROFISSIONAL E COMPLEMENTO

UPDATE PORTOPRINT.APOMAUTITM SET POFATVCPLCOD = NULL, POFATVCOD = NULL

WHERE POFATVCOD IS NOT NULL;

--DELETA COMPLEMENTO ATIVIDADE PROFISSIONAL

DELETE FROM PORTOPRINT.APOMPOFATVCPL;

### Shell script (.sh)

**O shell de execução**, tem como parâmetros que devem ser atualizados a cada atualitab:

[0] -> nomeArquivoQueries - Obrigatório

[1] -> flagAtomicidade - Opcional (DEFAULT = FALSE)

[2] -> flagIgnoraQtdadeLinhasAfetadasPorComando - Opcional (DEFAULT = FALSE)

#### Parametros na shell

**A flag de atomicidade** deve ser setada como TRUE quando os comandos são dependentes, se um dos comandos der erro, faz rollback em todos já efetuados e para a execução.

**A flagIgnoraQtdadeLinhasAfetadasPorComando**, deve ser setada como TRUE, quando um dos comandos da lista alterar mais de um registro por vez, exemplo:

-- Apaga todos registros de SUSEPs que estão como projeto piloto para o projeto grade.

DELETE FROM APOKSUSPRJLBR WHERE PILPRJCOD = 7;

**Os outros campos do script shell, devem continuar da mesma forma**.

### Arquivos de log

Os arquivos de logs são gerados automaticamente na pasta '/logeve' com o nome de '**nomeArquivoQueries.log**' para o log total e '**nomeArquivoQueries-err.log**' com os possíveis erros na execução dos comandos.

### Log4j – configuração assíncrona e segregação de log por JVM

Foram observados problemas de threads sendo bloqueadas em Produção no método org.apache.log4j.Category.callAppenders(...) do log4j. Este método contém um bloco synchronized que efetua a gravação do log no arquivo definido como appender. O arquivo de configuração do log4j define um único arquivo para registro de logs. Quando há mais de uma JVM na mesma máquina há grande concorrência de gravação neste arquivo visto que uma JVM desconhece a existência da outra, causando assim enfileiramento e lentidão na gravação dos logs.

Segue trecho do thread dump observado:

"http-localhost%2F127.0.0.1-8888-5" daemon prio=6 tid=0x37944000 nid=0x1a84 waiting for monitor entry [0x3910f000]

java.lang.Thread.State: BLOCKED (on object monitor)

at org.apache.log4j.Category.callAppenders(Category.java:203)

- waiting to lock <0x0eef0fc8> (a org.apache.log4j.spi.RootLogger)

Para diminuir a concorrência utilizamos uma extensão do log4j desenvolvida pela Magna Sistemas com o objetivo de segregar o log por JVM. O seguinte appender foi configurado:

***br.com.magnasistemas.log4j.ext.ClusterRollingFileAppender***

A gravação dos logs neste appender é feita pelo ***org.apache.log4j.AsyncAppender*** o qual foi configurado no log4j com um bufferSize de 200 eventos.

# Uso indiscriminado dos métodos get.

## Utilização dos métodos get/set.

**A geração de métodos get/set para o encapsulamento de atributos e amplamente difundida no mundo da orientação a objetos, sendo considerada uma prática vista como quase que obrigatória em muitos cenários.**

### Uso indiscriminado dos métodos get/set.

Mas nem tudo são flores quando estamos falando em encapsulamento, realmente tais métodos são muitas vezes necessários, mas será que são sempre necessários?

Toda vez que um atributo e acessado via get/set uma nova informação e adicionada na pilha podendo essa informação ser alocada na memória heap ou stack. Embora o garbage collector venha a deslocar esses objetos da memória devemos facilitar um pouco sua vida.

#### Cenário de uso

Devemos gerar os métodos get/set quando realmente necessário que tais atributos sejam acessados por outros objetos assim aplicando o encapsulamento, já dentro da classe Originaria de tais métodos não temos a necessidade de acessa-los através de tais métodos evitando assim que a pilha gerada na memória seja aumentada. Podemos abrir algumas exceções quando tais métodos conterem regras de negócios embora exista muitas controvérsias sobre o assunto.

**Link com referencias:**

http://blog.caelum.com.br/nao-aprender-oo-getters-e-setters/

http://www.viamais.net/blog/o-que-e-memoria-heap-e-stack-em-java/