



building enterprise applications in simple way.

Guia de Uso - Português

Autor

Ayslan Macedo

Sumário

1. Introdução	5
1.1. O que é o xpert-framework?	5
1.2. Xpert Sistemas	5
1.3. Sobre esta documentação	5
2. Padrão de documentação	6
3. Links úteis	7
4. Configuração do xpert-framework	8
4.1. Download	8
4.2. Dependência maven	8
4.3. Dependências	8
5. Configurando o arquivo xpert-config.xml	9
5.1. Visão Geral	9
5.2. Configurando o EntityManagerFactory	9
5.2.1. Introdução	9
5.2.2. Exemplo de implementação	9
6. Xpert-Faces	11
6.1. Introdução	11
6.2. Conversores	11
6.2.1. Entity Converter	11
6.3. Componentes JSF	11
6.3.1. Confirmation	11
6.3.2. Download	12
6.3.3. Initializer	13
6.3.4. Filter On Enter	13
6.3.5. Input Number	14
6.3.6. Spread Checkbox/Radio	14
6.3.7. Modal Messages	15
6.3.8. Find All Bean - Bean para consultas genéricas	15
6.3.9. BooleanSelectItens – Lista de SelectItens com valores booleanos	16
6.3.10. Legends – legenda para ações	16
6.3.11. DateFilter	17
7. Internacionalização de mensagens do BeanValidation	19
7.1. Introdução	19
7.2. Configuração	19
7.3. Como é feita a internacionalização	19
7.4. Tipos de validações suportadas	20
8. Artefatos padronizados	22
8.1. AbstractBusinessObject	22

8.1.1. Estrutura de um BusinessObject	22
8.1.2. Principais métodos da classe	23
8.2. AbstractManagedBean	24
8.2.1. Estrutura de um ManagedBean	24
8.2.2. Principais métodos da classe	25
9. Tratamento de Regra de Negócio com o Business Exception	27
9.1. Validação Básica	27
9.2. Validação com Múltiplas regras	27
9.3. Passagem de Parâmetros na mensagem	28
10. Exibindo mensagem com o FacesMessageUtils	29
10.1. Utilização Básica	29
10.2. Passagem de Parâmetros na mensagem	29
10.3. Business Exception em um FacesMessagesUtils	29
11. DAO Genérico - BaseDAO	30
11.1. Introdução	30
11.2. Métodos do BaseDAO	30
11.2.1. find	30
11.2.2. listAll	30
11.2.3. list	30
11.2.4. unique	31
11.2.5. count	31
11.2.6. listAttributes	31
11.2.7. findAttribute	31
11.2.8. findList	31
11.2.9. getInitialized	32
11.2.10. delete	32
11.2.11. remove	32
11.2.12. Query personalizada	32
11.3. Passando Parâmetros nos métodos do BaseDAO	33
11.3.1. Map<String, Object>	33
11.3.2. Restrictions(List<Restriction>) e Restriction	33
12. DataTable paginado no banco com o LazyDataModelImpl	36
12.1. Utilização Básica	36
12.2. Adicionar restrições na consulta do LazyDataModel	36
12.3. Filtros da coluna	36
13. Unicidade dos campos com UniqueField	37
13.1. Definindo campos únicos em um BO genérico	37
13.2. Customização de mensagem no UniqueField	37
13.3. Validação fora de um BO Genérico	38
14. Criação de relatórios com o FacesJasper	39

15. Geração de Código (CRUD) com o Xpert-Maker	40
15.1. Introdução	40
15.2. Utilização Básica via componente Swing	40
15.3. Utilização Básica via componente JSF	45
15.4. Estrutura de um CRUD gerado	48
15.4.1. Artefatos criados para cada Entidade	48
15.4.2. Artefatos únicos	49
15.4.3. Padrão da view para listagem de Registros – list{entidade}.xhtml	49
15.4.4. Padrão da view para detalhamento do registro – detail{entidade}.xhtml	51
15.4.5. Padrão da view para criação e edição do registro – create{entidade}.xhtml	51
15.5. Atributos mapeados e seus respectivos componentes na View	53
16. Auditando as entidades com o Xpert-Audit	54
16.1. Introdução	54
16.2. AbstractAuditng - Classe que representa uma auditoria feita	54
16.2.1. Introdução	54
16.2.2. Atributos	55
16.2.3. Configuração	55
16.2.4. Exemplo de uma implementação	55
16.3. AbstractMetadata - Classe que representa os metadados	56
16.3.1. Introdução	56
16.3.2. Atributos	57
16.3.3. Configuração	57
16.3.4. Exemplo de uma implementação	57
16.4. Listener para a auditar um objeto	58
16.4.1. Introdução	58
16.4.2. Configuração	58
16.4.3. Exemplo de uma implementação	58
16.5. Auditando uma entidade	59
16.5.1. Auditar um Insert	59
16.5.2. Auditar um Update	59
16.5.3. Auditar um Delete	59
16.5.4. BaseDAOImpl do xpert-framework e sua auditoria	59
16.6. Exibindo a auditoria de uma determinada entidade	60
16.7. Ignorar auditoria de uma classe ou de um atributo	60
17. Controle de Acesso através do xpert-security	62
17.1. Introdução	62
17.2. Gerenciamento da Sessão a partir da classe SecuritySessionManager	62
17.3. Bean para manter usuário e permissões na sessão	63
17.4. Filtro para bloquear o acesso do usuário por página	64
17.5. Bean para realizar login/logout do usuário na aplicação	65
17.6. Login utilizando SecurityLoginBean	67

17.7. SecurityArea para verificação de acesso a nível de componente	67
18. Arquétipo maven EAR do xpert-framework	68
18.1. Criando um projeto a partir de um arquétipo maven com Netbeans	68
18.2. Estrutura de um projeto gerado a partir do arquétipo maven	71
18.2.1. Main	71
18.2.2. EJB	72
18.2.3. WAR	72

1. Introdução

1.1. O que é o xpert-framework?

O xpert-framework surgiu com o objetivo de facilitar o desenvolvimento com JavaEE. Ao longo da experiência prática de desenvolvimento de projetos grandes surgiram problemas e a necessidade de alguns incrementos que a especificação do JavaEE e suas implementações não fornecem.

Além de JavaEE, o xpert-framework é fortemente ligado ao framework hibernate e ao primefaces, por isso ele também provê utilitários nessas tecnologias.

Esse framework pode ser considerado como um conjunto de componentes e utilitários, pois o núcleo do desenvolvimento nos projetos que utilizam ele continua sendo a linguagem java e especificações da tecnologia JavaEE.

1.2. Xpert Sistemas

A Xpert Sistemas é uma empresa brasileira de desenvolvimento de software e consultoria especializada em JavaEE, localizada em Teresina, Piauí.

1.3. Sobre esta documentação

Esta documentação trata da versão 1.4 do xpert-framework. O objetivo dela é tratar de maneira detalhada aquilo que o framework disponibiliza.

Durante os capítulos seguintes serão exibidos exemplos de códigos e utilização do que o framework fornece. São feitas menções ao javaee, pois o framework se baseia fortemente nelas então para dúvidas sobre a especificação javaee ou outra tecnologia como hibernate ou primefaces é aconselhável consultar a documentação das respectivas tecnologias.

Alguns exemplos podem ser vistos no showcase do framework, o link para ele está disponível a sessão links úteis.

2. Padrão de documentação

A documentação das classes, páginas e outros itens segue o seguinte padrão:

@Stateless – *Indica o Artefato*
JavaEE – *Indica a tecnologia*
Indica um EJB que não guarda estado – *Descrição do Artefato*

Para códigos fonte o cor da linha será destacada em cinza da seguinte maneira:

```
public void validate(Pessoa pessoa) throws BusinessException {  
    //carga horaria nao pode ultrapassar 9h  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("pessoa.business.cargaHorarioAcima");  
    }  
}
```

3. Links úteis

Showcase do xpert-framework

Exemplos de funcionamento dos componentes são disponíveis aqui.

<http://showcase.xpertsistemas.com.br/>

Homepage do xpert-framework no google codes

Atualmente o código fonte do framework e do showcase está disponível no google codes.

No google codes ainda é possível acompanhar o andamento de desenvolvimento, cadastrar bugs e sugerir melhorias.

<http://code.google.com/p/xpert-framework/>

Hibernate

<http://www.hibernate.org/>

Primefaces

<http://www.primefaces.org/>

4. Configuração do xpert-framework

4.1. Download

O xpert-framework é disponível através de um único jar que pode ser baixado através do link <https://code.google.com/p/xpert-framework/downloads/list>.

4.2. Dependência maven

Para adicionar a dependência o groupId é *com.xpert* e o artefato é *xpert-framework*:

```
<dependency>
  <groupId>com.xpert</groupId>
  <artifactId>xpert-framework</artifactId>
  <version>1.4</version>
</dependency>
```

O artefato pode ser encontrado no link indicado na sessão “Download”.

4.3. Dependências

Algumas dependências são necessárias para o xpert-framework, elas estão listadas abaixo:

- commons-beanutils
- commons-io
- commons-collections
- freemarker
- primefaces 3.x
- javaee-api
- hibernate 4.x
- JasperReports (apenas se for utilizar a classe FacesJasper)

Muitos dos componentes visuais do xpert-framework utilizam o primefaces 3.x, por isso é importante o projeto possuí-lo, para Classes utilitárias e outros componentes não visuais sua dependência não é necessária.

É importante ressaltar que apenas a versão 4.x do hibernate é suportada, isso ocorre devido a mudança da estrutura da versão 3 para a 4, um exemplo disso é a mudança de nome de alguns pacotes.

5. Configurando o arquivo xpert-config.xml

5.1. Visão Geral

Esse arquivo é necessário para fazer a configuração de alguns módulos e componentes do framework.

O xpert-config.xml deve está localizado no diretório **WEB-INF** do projeto.

Exemplo do xpert-config utilizado no showcase:

```
<xpert-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <auditing>
    <auditing-impl>com.xpert.showcase.audit.Auditing</auditing-impl>
    <metadata-impl>com.xpert.showcase.audit.Metadata</metadata-impl>
    <auditing-listener>com.xpert.showcase.session.AuditingListenerImpl</auditing-listener>
  </auditing>
  <entity-manager-factory>com.xpert.showcase.application.EntityManagerFactoryImpl</entity-manager-factory>
  <resource-bundle>bundles.messages</resource-bundle>
</xpert-config>
```

A seção **auditing** configura o módulo de auditoria, onde se indica a classe de auditoria (**auditing-impl**) e a classe de metadados (**metadata-impl**) e o listener para a auditoria (**auditing-listener**).

A tag **entity-manager-factory** define a classe que implementa **com.xpert.EntityManagerFactory**, que é necessária para operações onde se acessa o banco de dados.

A tag **resource-bundle** define o resource bundle a ser utilizado para a internacionalização do projeto.

5.2. Configurando o EntityManagerFactory

5.2.1. Introdução

Para a utilização de alguns componentes e classes utilitários do framework que fazem acesso ao banco de dados deve ser definido uma implementação da classe **com.xpert.EntityManagerFactory**, e ela deve ser colocada no xpert-config.xml, como mostrado na seção anterior.

Essa implementação só é importante caso se utilize componentes de acesso ao banco de dados, como o **initializer**, **componentes de auditoria** ou o **componente jsf do xpert-maker**.

5.2.2. Exemplo de implementação

Para esse exemplo vai ser utilizado a maneira onde se obtém o entity manager através de configuração no **web.xml**, outras implementações podem ser utilizadas também, basta que essa classe retorne o **EntityManager**.

Para esse exemplo é importante destacar que o arquivo **persistence.xml** deve ser visível para o módulo war (que possui o web.xml).

Implementação do EntityManagerFactory

```
import com.xpert.EntityManagerFactory;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.persistence.EntityManager;

public class EntityManagerFactoryImpl implements EntityManagerFactory {

    private static final Logger logger = Logger.getLogger(EntityManagerFactoryImpl.class.getName());
    private static final String ENTITY_MANAGER_REF_NAME = "java:comp/env/persistence/entityManager";

    @Override
    public EntityManager getEntityManager() {
        InitialContext ctx;
        try {
            ctx = new InitialContext();
            EntityManager entityManager = (EntityManager) ctx.lookup(ENTITY_MANAGER_REF_NAME);
            return entityManager;
        } catch (NamingException ex) {
            logger.log(Level.SEVERE, null, ex);
        } catch (Exception ex) {
            logger.log(Level.SEVERE, null, ex);
        }
        return null;
    }
}
```

Referência do entity manager no web.xml

Essa referência é indicada seguindo o padrão do javaee. O código abaixo deve ser adicionado no **web.xml**, sendo que o **persistence-unit-name** deve indicar o Persistence Unit que está no arquivo de configuração do JPA **persistence.xml**.

```
<persistence-context-ref>
  <persistence-context-ref-name>persistence/entityManager</persistence-context-ref-name>
  <persistence-unit-name>xpertShowcasePU</persistence-unit-name>
</persistence-context-ref>
```

Configurar xpert-config

Deve ser indicado a classe de implementação do EntityManagerFactory no arquivo xpert-config através da tag **entity-manager-factory**.

6. Xpert-Faces

6.1. Introdução

O objetivo desse módulo é prover artefatos utilitários para facilitar o uso da tecnologia Java Server Faces 2.0 (JSF), para isso, o xpert-framework disponibiliza componentes, conversores e classes utilitárias.

Os componentes pode ser importados a partir dos seguintes namespaces:

- `http://xpert.com/faces`
- `http://java.sun.com/jsf/composite/xpert/components` (para composite components)

Exemplo:

```
<html xmlns:x="http://xpert.com/faces"
      xmlns:xc="http://java.sun.com/jsf/composite/xpert/components">

</html>
```

6.2. Conversores

6.2.1. Entity Converter

Esse é um conversor genérico para entidades JPA, seguindo sua especificação. A conversão não limita apenas a essas entidades, sendo possível converter outros objetos, onde seja criado algum identificador e seu atributo/método seja anotado com `@ConverterId` do pacote `com.xpert.faces.conversion.ConverterId`.

É importante destacar que para seu funcionamento a classe deve possuir seu devido método equals sendo feito pelo identificador da entidade.

Abaixo um exemplo de sua utilização:

```
<h:selectOneMenu value="#{bean.group}" converter="entityConverter" >
  <f:selectItems value="#{bean.groups}" var="group"
    itemLabel="#{group.description}" />
</h:selectOneMenu>
```

O funcionamento desse conversor se dá da seguinte maneira, cada objeto da lista é adicionado ao ViewMap do JSF, com isso ao submeter a página o objeto é recuperado desse Map. Isso torna desnecessário consultas extras no banco de dados ou a criação de um conversor para cada entidade.

6.3. Componentes JSF

6.3.1. Confirmation

Componente para exibir uma confirmação da ação feita a partir de um `commandButton` ou `commandLink`. Sua idéia é semelhante a de um “confirm” do javascript, porém integrado com os componentes `command ajax` e `não ajax`.

Atributos

Nome	Valor Padrão	Tipo	Descrição
message	Confirma?	String	Define a mensagem a ser exibida de confirmação
confirmLabel	Sim	String	Define a valor do botão de confirmação.
cancelLabel	Não	String	Define o valor do botão da não confirmação

Utilização

Usando commandButton do JSF com ajax:

```
<h:commandButton value="Submit" >
  <f:ajax render="@form" execute="@form"/>
  <x:confirmation/>
</h:commandButton>
```

Usando commandButton do Primefaces e definindo uma mensagem customizada:

```
<p:commandButton process="@form" update="@form" value="Submit (With custom message)" >
  <x:confirmation message="Are you sure?" confirmLabel="Of course" cancelLabel="No way"/>
</p:commandButton>
```

Ao se utilizar o commandLink do primefaces, devido ao seu comportamento não é possível utilizar o confirmation. Pelos testes realizados, quando se adiciona algum componente a ele, além do confirmation, então o confirmation é renderizado.

6.3.2. Download

Componente para bloquear a tela e adicionar eventos quando a requisição for o download de algum arquivo. Ele pode ser utilizado com commandButton, commandLink do JSF ou Primefaces. Sendo para o primefaces assim como o componente *confirmation* o commandLink apresenta um comportamento diferente.

Para gerar o download deve ser chamado o método *FacesUtils.download()*.

Atributos

Nome	Valor Padrão	Tipo	Descrição
showModal	true	Boolean	Define a tela deve ser bloqueada enquanto a requisição é completa
message	Carregando	String	Define a mensagem a ser exibida enquanto a tela é bloqueada.
onstart		String	Javascript a ser executado ao iniciar a requisição
oncomplete		String	Javascript a ser executado ao finalizar a requisição

Utilização

Usando commandButton do JSF:

```
<h:commandButton action="#{downloadMB.download}" value="Download">
  <x:download/>
</h:commandButton>
```

DownloadMB:

```
public class DownloadMB {

    public void download() throws IOException, InterruptedException {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(byteArrayOutputStream));
        //wait 5s
        Thread.sleep(5000);
        writer.close();
        FacesUtils.download(byteArrayOutputStream.toByteArray(), "text/plain", "arquivo.txt");
    }
}
```

6.3.3. Initializer

Componente para inicializar os objetos Lazy do Hibernate/JPA na view. Este componente evita o famoso *LazyInitializationException* ao fazer referência a um objeto *lazy* na view.

Utilização

```
<h:outputText value="#{bean.entity.lazyAttribute.description}" >
  <x:initializer/>
</h:outputText>
```

Para recuperar o entity manager e inicializar o objeto o xpert-framework precisa que seja definido o EntityManagerFactory e que seja devidamente configurado, isso pode ser visto na sessão [Configurando o EntityManagerFactory](#).

6.3.4. Filter On Enter

Componente para limitar a consulta do componente *dataTable* do primefaces quando se utiliza a opção *filterBy* da coluna a ser realizada apenas quando se aperta o *Enter*.

Atributos

Nome	Valor Padrão	Tipo	Descrição
target		String	Id da tabela a ser utilizado o filterOnEnter.
selector		String	Seletor jQuery para retornar as tabelas a serem utilizadas.

Utilização

Quando os atributos target e selector não são definidos, por padrão todas as tabelas serão aplicadas a regra.

```
<p:dataTable id="dataTablePerson" var="person" value="#{personMB.dataModel}">
  <p:column headerText="#{msg['person.name']}" sortBy="#{person.name}"
    filterBy="#{person.name}">
    <h:outputText value="#{person.name}"/>
  </p:column>
```

```
</p:dataTable>  
<x:filterOnEnter/>
```

6.3.5. Input Number

Componente para inserir mascara para valores numéricos. Útil por exemplo para valores monetários.

Atributos

Possui basicamente os mesmos atributos de um `inputText`, adicionando os seguinte componentes:

Nome	Valor Padrão	Tipo	Descrição
<code>allowNegative</code>	<code>false</code>	Boolean	Indica se podem ser informados valores negativos (deve-se pressionar a tecla ‘-’)
<code>limit</code>	<code>15</code>	Integer	Quantidade limite de casas numéricas (decimais e interias)
<code>centsSeparator</code>	Conforme o Locale	String	Separador de centésimos
<code>thousandsSeparator</code>	Conforme o Locale	String	Separador de milhares
<code>centsLimit</code>	<code>2</code>	Integer	Quantidades limite de números não inteiros

Utilização

Utilização Básica:

```
<xc:inputNumber value="#{bean.value}" />
```

Aceitando números negativos:

```
<xc:inputNumber value="#{bean.value}" allowNegative="true"/>
```

6.3.6. Spread Checkbox/Radio

Componente para quebrar em colunas os components `h:selectManyCheckBox` e `h:selectManyRadio`.

Esses componentes são limitados a dizer apenas se são na horizontal e na vertical, com o componente `spread` é possível determinar a quantidade de colunas deles. Este componente adiciona ainda um estilo CSS para o `background` do `checkbox` marcado mudar a cor.

Atributos

Nome	Valor Padrão	Tipo	Descrição
<code>columns</code>		Integer	Quantidade de colunas
<code>highlight</code>	<code>true</code>	Boolean	Indica se o estilo deve ser aplicado ou não nos itens selecionados

Utilização

Utilização básica:

```
<h:selectManyCheckbox >  
  <x:spread columns="3"/>  
  <f:selectItems value="#{bean.items}" />  
</h:selectManyCheckbox>
```

6.3.7. Modal Messages

Componente para exibição de mensagens do JSF dentro de um dialog. É um composite componente que utiliza os componentes *dialog* e *messages* do primefaces.

Atributos

Nome	Valor Padrão	Tipo	Descrição
autoUpdate	false	Boolean	Indica se o componente deve se atualizar na requisição ajax

Utilização

Utilização básica:

```
<xc:modalMessages/>
```

6.3.8. Find All Bean - Bean para consultas genéricas

6.3.8.1. Introdução

Para consultas genéricas o xpert-framework disponibiliza a classe FindAllBean, através dela é possível fazer consultas das entidades mapeadas.

Ele é muito útil por exemplo na geração de código do módulo xpert-maker, onde para os atributos mapeados com @ManyToOne ou @ManyToMany são criados componentes h:selectOne, e eles já são preenchidos do banco de dados.

6.3.8.2. Exemplo de implementação

O exemplo a seguir mostra um ManagedBean do scope de visão, e nele é mapeado a classe Group. Ao criar uma classe filha dela é necessário sobrescrever o método getClassModel.

Esse classModel pode ser usado para indicar a ordenação da entidade recuperada, nesse caso **group** será ordenado pelo atributo **description**.

```
@ManagedBean
@ViewScoped
public class FindAllBean extends com.xpert.faces.bean.FindAllBean {

    private static final Map<Class, ClassModel> MODEL = new HashMap<Class, ClassModel>();

    static {
        MODEL.put(Group.class, new ClassModel("description"));
    }

    @Override
    public Map<Class, ClassModel> getClassModel() {
        return MODEL;
    }
}
```


No classModel é possível ainda indicar o label a ser utilizado na exibição quando o retorno é uma lista do objeto SelectItem. O retorno SelectItem é utilizado por exemplo do atributo *filterOptions* da coluna do **dataTable** do primefaces.

6.3.8.3. Exemplo de utilização

Pode ser utilizado para preencher combobox na tag selectItems:

```
<h:selectOneMenu value="#{bean.country}" converter="entityConverter" >
  <f:selectItems value="#{findAllBean.get(classMB.country)}" var="country"
    itemLabel="#{country.name}" />
</h:selectOneMenu>
```

Pode ser utilizado no *filterOptions* do dataTable do primefaces, sendo que para esse caso deve retornar uma lista de select item e deve-se utilizar o método **getSelect**:

```
<p:column headerText="#{msg['person.status']}" sortBy="#{person.status}"
  filterBy="#{person.status}"
  filterOptions="#{findAllBean.getSelect(classMB.status)}">
  <h:outputText value="#{person.status}" />
</p:column>
```

Muito da geração de código fornecida pelo xpert-maker faz referência a esse findAllBean, e ele deve ser considerado apenas como um utilitário genérico, lógicas mais complexas de recuperação de dados não devem ser feitas nele.

6.3.9. BooleanSelectItems – Lista de SelectItems com valores booleanos

Esse managed bean é útil para se recuperar os valores booleanos quando necessários em uma página.

BooleanSelectItems

Acessível a través da EL **#{booleanSelectItems}** Exibe apenas as opções true ou false:

```
<h:selectOneRadio>
  <f:selectItems value="#{booleanSelectItems}" />
</h:selectOneRadio>
```

BooleanSelectItemsEmptyOption

Acessível a través da EL **#{booleanSelectItemsEmptyOption}** Exibe as opções true, false e uma opção vazia, ela é muito útil para se carregar as opções de um filtro da coluna do primefaces:

```
<p:column headerText="Boolean Value" sortBy="#{object.booleanValue}"
  style="text-align: center;" filterBy="#{object.booleanValue}"
  filterOptions="#{booleanSelectItemsEmptyOption}">
  <h:outputText value="#{object.booleanValue}" converter="yesNoConverter" />
</p:column>
```

6.3.10. Legends – legenda para ações

Composite Componente para exibir uma lista de comandos possíveis. Útil por exemplo para colocar um legenda no data table, especificando os botões “detalhar”, “editar” e “exclusão”.

Atributos

Possui basicamente os mesmos atributos de um `inputText`, adicionando os seguinte componentes:

Nome	Valor Padrão	Tipo	Descrição
vertical	false	Boolean	Indica se os itens devem ser dispostos na vertical, por padrão são 2 colunas.
edit	false	Boolean	Exibir item "Editar"
detail	false	Boolean	Exibir item "Detalhar"
delete	false	Boolean	Exibir item "Exclusão"
download	false	Boolean	Exibir item "Download"
report	false	Boolean	Exibir item "Download"
select	false	Boolean	Exibir item "Selecionar"
print	false	Boolean	Exibir item "Imprimir"
replace	false	Boolean	Exibir item "Substituir"

Utilização

Utilização básica:

```
<xc:legends detail="true" edit="true" delete="true"/>
```

Legendas na posição vertical:

```
<xc:legends detail="true" edit="true" delete="true" vertical="true"/>
```

Como cabeçalho de uma coluna do `dataTable`:

```
<p:column>
  <f:facet name="header">
    <xc:legends detail="true" edit="true" delete="true"/>
  </f:facet>
</p:column>
```

6.3.11. DateFilter

Este component facilita o uso de filtros com campos de data no `dataTable` que utilize o `LazyDataModelImpl`.

Esse componente renderiza 2 componentes calendar do primefaces, um sendo a data inicial da consulta e outro sendo a data final. Ele funciona por causa do `RestrictionType` "DATA_TABLE_FILTER" e tem um tratamento diferenciado, ao selecionar a data de início da consulta é adicionado a restrição "GREATER_EQUALS_THAN" e ao se selecionar a data final é adicionado a restrição "GREATER_LESS_THAN".

Para ser renderizado ele deve ser definido no facet “header” do componente “column”.

Utilização

```
<p:column sortBy="#{object.date}"
    filterBy="#{object.date}" style="text-align: center;">
  <f:facet name="header">
    Date Field
    <x:dateFilter/>
  </f:facet>
  <h:outputText value="#{object.date}"/>
</p:column>
```

7. Internacionalização de mensagens do BeanValidation

7.1. Introdução

A api 2.0 do jsf trouxe integração automática com a especificação BeanValidation (JSR 303). Essa integração facilitou muito a utilização das anotações do bean validation em projetos que utilizam JSF.

O problema dessa integração é que as mensagens não são formatadas para um determinado atributo, por exemplo, uma classe `Person` com o atributo `name` anotado com `@NotNull`, o jsf exibiria a mensagem “value is required”, ou seja, é uma mensagem genérica.

Nesse exemplo o ideal seria exibir “Name is required” e melhor ainda exibir “Nome é obrigatório” no caso do Locale `pt_BR`.

Diante desse problema o `xpert-framework` disponibiliza um *Interpolator* próprio para tratar essas mensagens, sendo que essa classe é uma implementação da classe `javax.validation.MessageInterpolator` da api do java-ee e uma implementação do `BeanValidator` que é filha de `javax.faces.validator.BeanValidator`.

7.2. Configuração

Para indicar o interpolator do `xpert-framework` deve ser criado o arquivo `validation.xml` que é um arquivo de configuração próprio da api validation do java-ee dentro da pasta `META-INF` e indicar a classe `com.xpert.i18n.CustomInterpolator`.

Exemplo:

```
<validation-config xmlns="http://jboss.org/xml/ns/javax/validation/configuration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://jboss.org/xml/ns/javax/validation/configuration validation-configuration-1.0.xsd">
  <message-interpolator>com.xpert.i18n.CustomInterpolator</message-interpolator>
</validation-config>
```

Com o interpolator configurado, é necessário indicar o resource bundle a ser utilizado pelo `xpert framework`, ele pode ser indicado através do `xpert-config.xml` na seção **resource-bundle**.

Para que o jsf reconheça o novo validador é necessário sobrescrever no `faces-config.xml` o `BeanValidator`:

```
<validator>
  <validator-class>com.xpert.core.validation.BeanValidator</validator-class>
  <validator-id>javax.faces.Bean</validator-id>
</validator>
```

7.3. Como é feita a internacionalização

O nome do campo se baseia na seguinte lógica:

Nome simples da classe concatenado com “**. (ponto)**” concatenado com o **nome do atributo**.

O nome da classe deve iniciar com minúsculo (*lower camel case*).

Considerando a classe Person abaixo o com atributo name:

```
@Entity
public class Person {

    @NotBlank
    private String name;

}
```

O arquivo de internacionalização ficaria:

pt_BR (Português do Brasil)

person.name=Nome

en (Inglês)

person.name=Name

Assim a mensagem ficaria:

Nome é Obrigatório

Para colocar uma mensagem personalizada basta adicionar o atributo message na anotação. Exemplo:

```
@Entity
public class Person{

    @NotBlank(message="Name is required. This is a custom message")
    private String name;

}
```

7.4. Tipos de validações suportadas

Algumas das validações da especificação são suportadas e além delas algumas do hibernate-validator também. Algumas validações valores podem ser definidos, como é o caso do @Size onde pode-se indicar o min e o max, a mensagem gerada seria algo do tipo “Valor deve possuir no máximo \${min} caracteres e no máximo {max}”

Java-ee api

- NotNull
- Max
- Min
- Size
- DecimalMax
- DecimalMin
- Past
- Future

Hibernate validator

- NotBlank
- NotEmpty
- Email
- Range
- URL

Para mais informações sobre cada tipo de validação é importante consultar a especificação, pois a parte de validação é feita a através dela e suas específicas implementações e o xpert-framework fica a cargo apenas a formatação das mensagens.

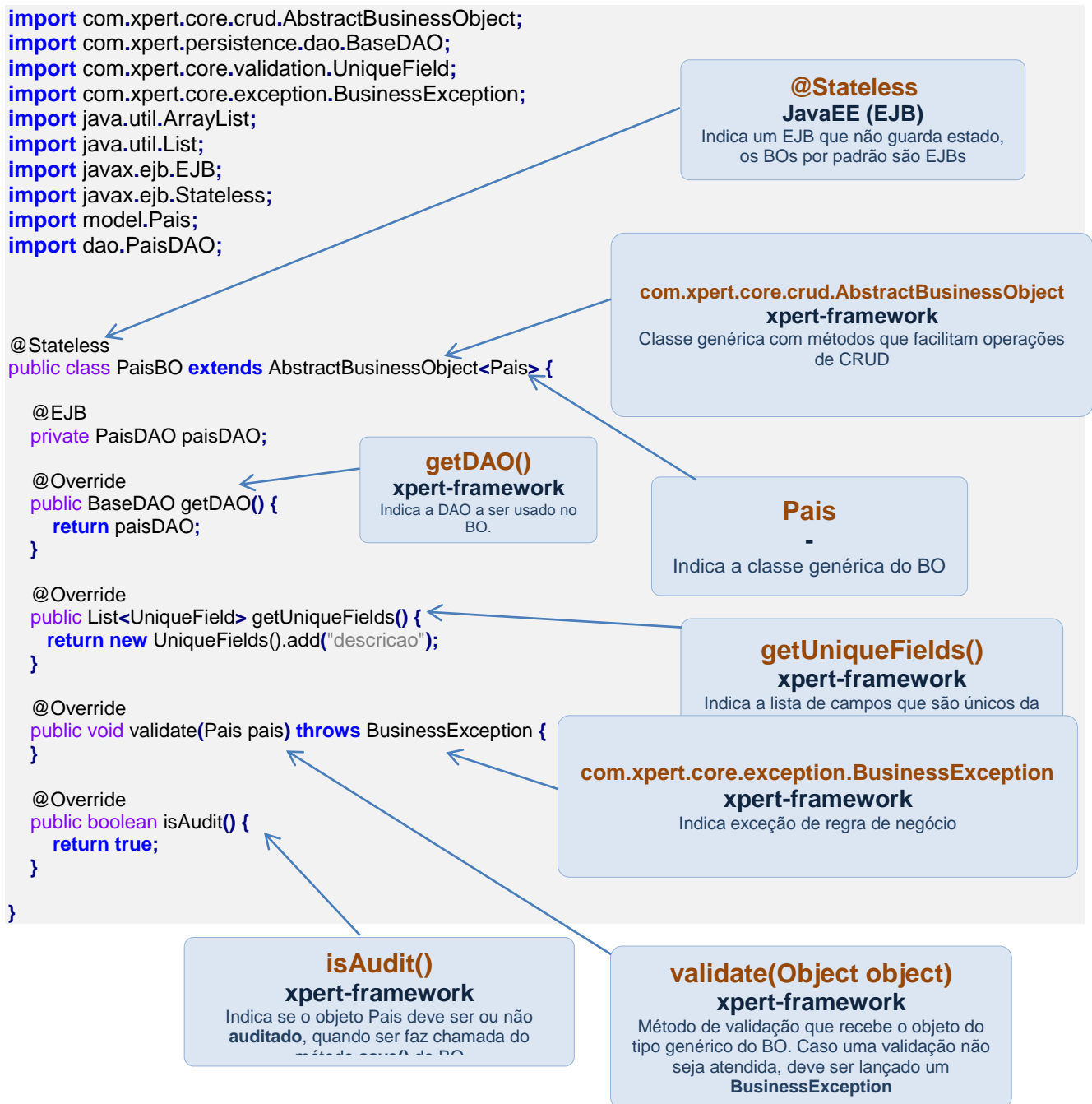
8. Artefatos padronizados

8.1. AbstractBusinessObject

Classe genérica de regra de negócio para facilitar a Criação de CRUDs.

8.1.1. Estrutura de um BusinessObject

Exemplo de um BO para País:



8.1.2. Principais métodos da classe

8.1.2.1. *getUniqueFields* – Definir Unicidade dos campos

Este Método retorna uma lista de UniqueField com os campos que não devem ser repetidos. Exemplo: um Estado de um País não pode conter descrição ou siglas repetidas.

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields()
        .add("descricao", "pais")
        .add("sigla", "pais");
}
```

8.1.2.2. *isAudit* – Definir se a entidade deve ser auditada

Este método define se a entidade deve ser auditada ao se chamar o método **save** do BO.

```
@Override
public boolean isAudit() {
    return true;
}
```

8.1.2.3. *validate* – Validação simples de uma entidade

Este método recebe a entidade do BO e é possível se fazer validação lançando um BusinessException
Exemplo:

```
@Override
public void validate(Pessoa pessoa) throws BusinessException {
    if (pessoa.getPerfil() == null) {
        throw new BusinessException("required.perfil");
    }
}
```

Este método recebe apenas a própria entidade, caso seja necessário passar mais objetos para a validação, deve ser feita uma sobrecarga do método, e para que o novo método criado seja chamado é necessário sobrescrever o método **save()**.

8.1.2.4. *save* – Método para persistir a Entidade

O método **save** da classe abstrata obedece o seguinte fluxo:

1. Chamar o método **validade()**
2. Chamar o método **validateUniqueFields()** (validação os campos definidos no **getUniqueFields()**)
3. Lançar exceção caso exista problemas na validação
4. Chamar **persist** caso seja um novo objeto, ou **merge** para atualizar um objeto do DAO.

Para mudar esse fluxo ou chamar algum validate específico, o método **save()** deve ser sobrescrito, ou ainda sobrecarregado.

8.2. AbstractManagedBean

Managed Bean genérico para criação de CRUDs.

8.2.1. Estrutura de um ManagedBean

Exemplo de um ManagedBean para País:

```
import com.xpert.core.crud.AbstractBaseBean;  
import com.xpert.core.crud.AbstractBusinessObject;  
import javax.ejb.EJB;  
import javax.faces.bean.ManagedBean;  
import javax.faces.bean.ViewScoped;  
import bo.PaisBO;  
import model.Pais;
```

```
@ManagedBean  
@ViewScoped  
public class PaisMB extends AbstractBaseBean<Pais> {  
  
    @EJB  
    private PaisBO paisBO;  
  
    @Override  
    public AbstractBusinessObject getBO() {  
        return paisBO;  
    }  
  
    @Override  
    public String getDataModelOrder() {  
        return "descricao";  
    }  
}
```

com.xpert.core.crud.AbstractBaseBean
xpert-framework

Classe genérica com métodos que facilitam operações de CRUD

@ManagedBean
JavaEE (JSF)

Indica que a classe é um bean gerenciado do JSF.

@EJB
JavaEE (EJB)

Indica que o atributo é uma injeção de dependência do EJB.

getBO()
xpert-framework

Indica o BO a ser utilizado por esse ManagedBean.

@ViewScoped
JavaEE (JSF)

Indica que o managedbean é do escopo de View. Outras possibilidades são @SessionScoped e @RequestScoped (é o padrão quando não se usa anotação)

getDataModelOrder()
xpert-framework

Define a ordenação padrão de listagem do objeto. Nesse exemplo, a lista de países será ordenada por **descricao**

8.2.2. Principais métodos da classe

8.2.2.1. *getDataModelOrder* – Definir ordenação padrão da Listagem

Este método define a ordenação na listagem do LazyDataModel (primefaces).

Exemplo - Ordenar pelo atributo *nome*:

```
@Override
public String getDataModelOrder() {
    return "nome";
}
```

Pode ainda ser definida se a ordenação é asc ou desc. Exemplo:

```
@Override
public String getDataModelOrder() {
    return "dataCadastro DESC";
}
```

8.2.2.2. *postConstruct* – Método chamado no evento *@PostConstruct*

O método ***postConstruct()*** é chamado obedecendo o seguinte fluxo:

1. Verificar se foi enviado o campo ***id*** nos parâmetros
2. Caso seja enviado ***id*** recuperar a entidade do banco e setar como entidade atual
3. Chamado o ***createDataModel()*** para criar o LazyDataModel da entidade.
4. Chamado o método ***init()***.

8.2.2.3. *init* – Método chamado após o evento *@PostConstruct*

Este método é chamado no *@PostConstruct*, após a entidade ser carregada.

8.2.2.4. *getDataModelRestrictions* – Restringir dados do LazyDataModel

Este método retorna uma lista de restrições que devem ser usadas ao se fazer a consulta para renderizar o dataTable da entidade.

Exemplo – retornar apenas os registros de um usuário:

```
@Override
public List<Restriction> getDataModelRestrictions() {
    Restrictions restrictions = new Restrictions();
    restrictions.add("usuario", sessaoUsuarioMB.getUser());
    return restrictions;
}
```

8.2.2.5. *preSave* e *postSave* - Chamar eventos antes e depois de salvar a entidade

O método ***preSave()*** é chamado antes de salvar a entidade, enquanto o ***postSave()*** após. Esses métodos podem ser utilizados para controle de tela, por exemplo, quando salvar a entidade x, carregar a lista y. Lógicas complexas devem ser evitadas nesses métodos, onde o mais aconselhável é sobrescrever o método ***save()*** para se ter um código mais legível.

8.2.2.6. create – Criar uma nova instância da entidade

O método **create()** cria uma nova instância do objeto **entity** do Managed Bean.

8.2.2.7. getOrderByHandler – Customizar Ordenação do dataTable no sortBy

Com este método é possível definir uma ordenação customizada ao se ordenar a coluna do dataTable.

Exemplo, ao clicar para ordenar a coluna **number** adicionar também a coluna **year** na ordenação:

```
<p:column headerText="#{msg['object.number']}" sortBy="#{object.number}">
  <h:outputText value="#{object.number}"/>
</p:column>
```

Veja que o sortBy da tabela está pelo campo **object.number**. Sendo assim é possível tratar essa ordenação no ManagedBean sobrescrevendo o método **getOrderByHandler** da seguinte forma:

```
@Override
public OrderByHandler getOrderByHandler() {
    return new OrderByHandler() {
        @Override
        public String getOrderBy(String orderBy) {
            if (orderBy != null && orderBy.equals("number")) {
                return "number, year";
            }
            return orderBy;
        }
    };
}
```

Dessa forma é possível tratar as mais diversas ordenações, e para cada campo fazer uma maneira personalizada.

8.2.2.8. setLoadEntityOnPostConstruct – definir se o objeto será carregado a partir do id passado na requisição

Por padrão este método é **true**, setando este método como false caso seja passado um id nos parâmetros (via GET) este não será carregado.

9. Tratamento de Regra de Negócio com o Business Exception

BusinessException (com.xpert.core.exception) é uma *Exception* utilizada para **Regras de Negócio**.

Considerando o arquivo de internacionalização:

```
business.cargaHorarioAcima=Carga horária não pode ser acima de 9h
business.dataNascimentoNaoPodeSerFutura=Data de nascimento não pode ser uma data futura
business.cargaHorariaNaoPermitida=Carga horária {0} não permitida
business.pessoaComCargaHorariaInvalida=Pessoa de nome {0} com a carga horaria inválida. Carga horária informada:
{1}
```

9.1. Validação Básica

Regras:

1. Uma pessoa não pode ter a carga horaria acima de 9h.

```
public void validate(Pessoa pessoa) throws BusinessException {
    //carga horaria nao pode ultrapassar 9h
    if (pessoa.getCargaHoraria() > 9) {
        throw new BusinessException("pessoa.business.cargaHorarioAcima");
    }
}
```

9.2. Validação com Múltiplas regras

Regras:

1. Uma pessoa não pode ter a carga horaria acima de 9h.
2. A data de nascimento não pode ser uma data futura

```
public void validateMultipleException(Pessoa pessoa) throws BusinessException {
    BusinessException exception = new BusinessException();

    if (pessoa.getCargaHoraria() > 9) {
        exception.add("pessoa.business.cargaHorarioAcima");
    }
    if (pessoa.getDataNascimento().after(new Date())) {
        exception.add("pessoa.business.datanascimentoNaoPodeSerFutura");
    }
    exception.check();
}
```

O método **exception.ckeck()** verifica se alguma exceção foi adicionada e lança a exceção.

9.3. Passagem de Parâmetros na mensagem

Algumas mensagens possuem parâmetros dinâmicos. Exemplo:

“Carga horária 10h não permitida.” (Note que 10h pode ser um parâmetro informado pelo usuário).

```
public void validate(Pessoa pessoa) throws BusinessException {  
    /*  
     * Mensagem no arquivo de internacionalizacao: Carga horária {0} não permitida.  
     * Esse parametro "{0}" espera um valor, no exemplo abaixo para uma carga horária 20 a mensagem ficaria:  
     *  
     * "Carga horária 20h não permitida"  
     *  
     */  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("pessoa.business.cargaHorariaNaoPermitida", pessoa.getCargaHoraria()+"h");  
    }  
  
    /*  
     * Podem ser passados múltiplos parametros, basta seguir a ordem dos parametros "{0}, {1}, {2}..."  
     * Uma pessoa de nome "Maria" e carga horaria informada de 10, a mensagem ficaria:  
     *  
     * "Pessoa de nome Maria com a carga horaria inválida. Carga horária informada: 10h"  
     *  
     */  
    if (pessoa.getCargaHoraria() > 9) {  
        throw new BusinessException("pessoa.business.pessoaComCargaHorariaInvalida", pessoa.getNome(),  
            pessoa.getCargaHoraria()+"h");  
    }  
}
```

10. Exibindo mensagem com o FacesMessageUtils

FacesMessageUtils (com.xpert.faces.utils) é uma classe de utilitários para exibição de mensagens na View.

10.1. Utilização Básica

Informação:

```
FacesMessageUtils.info("mensagem");
```

Warning:

```
FacesMessageUtils.warning("mensagem");
```

Erro:

```
FacesMessageUtils.error("mensagem");
```

Fatal:

```
FacesMessageUtils.fatal("mensagem");
```

10.2. Passagem de Parâmetros na mensagem

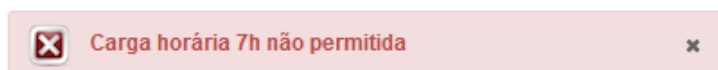
Considerando o arquivo de internacionalização:

```
peessoa.business.cargaHorariaNaoPermitida=Carga horária {0} não permitida
```

Chamada da mensagem:

```
FacesMessageUtils.error("peessoa.business.cargaHorariaNaoPermitida", 7+"h");
```

Resultado na view utilizando **p:messages** (Primefaces):



10.3. Business Exception em um FacesMessagesUtils

Um Business Exception pode ser passado para exibição no FacesMessagesUtils. Caso essa exception possua várias mensagens, o resultado renderizado será uma lista de mensagens.

```
public void save(Pessoa pessoa){  
    try {  
        validate(pessoa);  
    } catch (BusinessException ex) {  
        FacesMessageUtils.error(ex);  
    }  
}
```

11. DAO Genérico - BaseDAO

11.1. Introdução

O xpert-framework possui um conjunto de classes para a parte de Persistência, dentre eles o DAO genérico chamado **BaseDAO**, sendo que sua implementação é o **BaseDAOImpl** (ambos do pacote **com.xpert.persistence.dao**).

Os principais métodos do BaseDAO são:

- find
- listAll
- list
- unique
- count
- findAttribute
- listAttributes

11.2. Métodos do BaseDAO

11.2.1. find

Seleciona do banco uma entidade a partir do seu id.

Exemplo - Recuperar a Pessoa do id 100 (tipo Long):

```
Pessoa pessoa = pessoaDAO.find(100L);
```

Pode ser utilizado passando a classe:

```
Pessoa pessoa = dao.find(100L, Pessoa.class);
```

11.2.2. listAll

Seleciona do banco todos os registros de uma entidade.

```
List<Pessoa> pessoas = pessoaDAO.findAll();
```

Trazendo o resultado ordenado por nome:

```
List<Pessoa> pessoas = pessoaDAO.findAll("nome");
```

11.2.3. list

Retorna uma lista de registros a partir dos parâmetro informados.

Exemplo - Recuperar pessoas com carga horária de 8h, ordenados por **id**:

```
Restrictions restrictions = new Restrictions();  
restrictions.add("cargaHoraria", 8);  
pessoas = pessoaDAO.list(restrictions, "id");
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
pessoas = pessoaDAO.list("cargaHoraria", 8, "id");
```

11.2.4. unique

Semelhante ao *list()*, porém esse retorna apenas um único resultado.

Exemplo - Recuperar a pessoa de nome "JOHN".

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "JOHN");
pessoas = pessoaDAO.unique(restrictions);
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
pessoas = pessoaDAO.unique("nome", "JOHN");
```

O método *unique()* deve ser usado com cuidado, pois caso existam 2 resultados, ele limitará apenas a 1.

11.2.5. count

Retorna o total (*java.lang.Long*) de registros a partir dos parâmetro informados.

Exemplo - Recuperar total de pessoas com o nome carga horária de 8h, ordenados por **id**:

```
List< Restriction> restrictions = new ArrayList<Restriction>();
restrictions.add(new Restriction("cargaHoraria", 8));
Long total = pessoaDAO.count(restrictions);
```

O método ainda pode ser chamado da seguinte maneira e possuirá o mesmo resultado:

```
Long total = pessoaDAO.count("cargaHoraria", 8);
```

11.2.6. listAttributes

Recupera o objeto por demanda, informando apenas os campos a serem recuperados.

Supondo que existam 10 atributos no objeto Pessoa, desses atributos é necessário apenas o **nome** e o **id**, a chamada do método ficaria assim:

```
pessoas = pessoaDAO.listAttributes("id, nome");
```

Assim como os métodos citados anteriormente, é possível passar *Map* ou *Restrictions*, para filtrar a consulta.

11.2.7. findAttribute

Recupera algum atributo de um dado registro.

Supondo que você tenha em mãos o *id* de uma Pessoa e quer recuperar o perfil. Para evitar recuperar o objeto pessoa e depois fazer o *getPerfil()*, por questão de desempenho pode-se utilizar dessa maneira:

```
perfil = pessoaDAO.findAttribute("perfil", 1L);
```

Também é possível, passar o objeto no lugar do *id*:

```
perfil = pessoaDAO.findAttribute("perfil", pessoa);
```

11.2.8. findList

Semelhante ao ***findAttribute***, porém este retorna uma lista.

```
perfil = pessoaDAO.findList("permissoes", 1L);
```


Também é possível, passar o objeto no lugar do id:

```
perfil = pessoaDAO.findList("permissoes", pessoa);
```

11.2.9. getInitialized

Força recuperar um objeto LAZY carregado. Supondo que Pessoa possua o atributo perfil, se em um dado momento, perfil esteja *lazy*, acessar esse objeto causaria um

LazyInitializationException, para pegar esse atributo, uma das soluções seria usar o método **findAttribute()**, citado anteriormente, a outra seria usar **getInitialized()**, da seguinte maneira:

```
perfil = pessoaDAO.getInitialized(pessoa.getPerfil());
```

11.2.10. delete

Deleta a entidade a partir do id.

Exemplo - deletar a pessoa de id 11:

```
pessoaDAO.delete(11L);
```

O método delete(), não considera o cascade mapeado no JPA.

11.2.11. remove

Deleta a entidade a partir do objeto.

Exemplo - deletar uma determinada pessoa:

```
pessoaDAO.remove(pessoa);
```

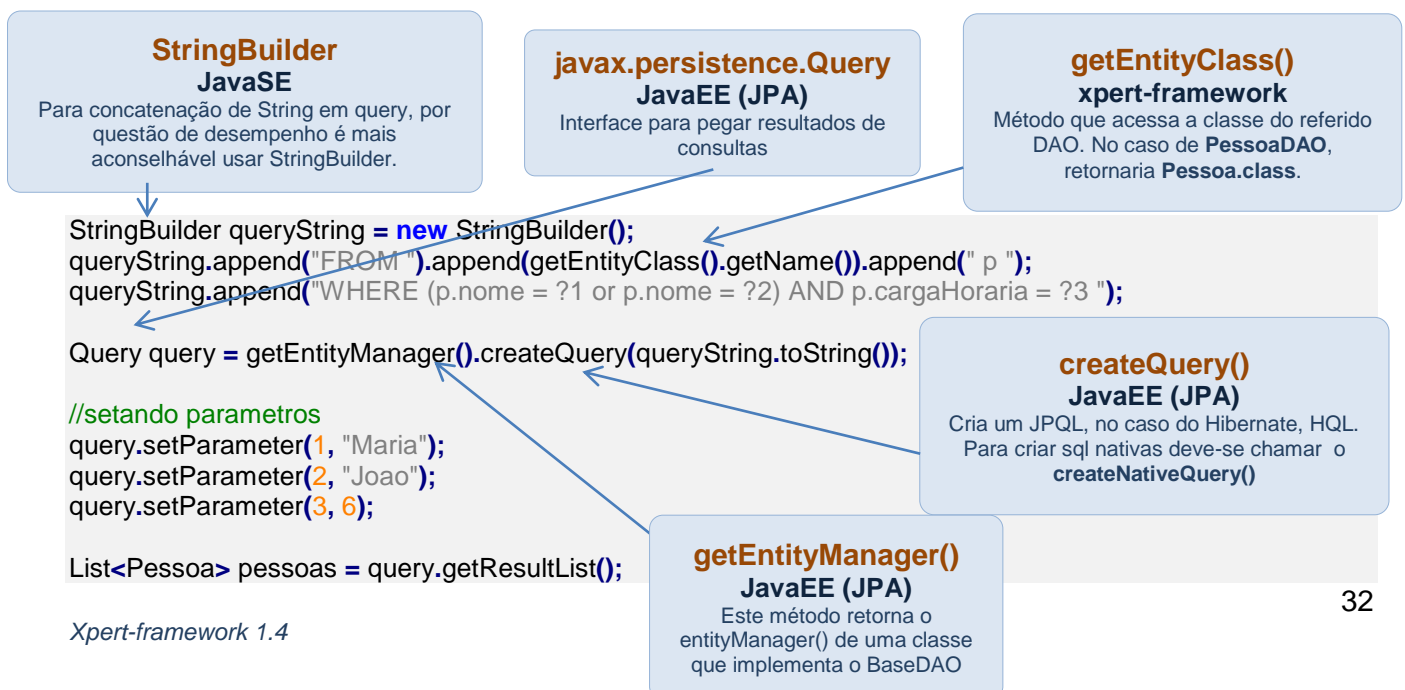
O método remove() faz chamada ao **entityManager.remove()** do JPA, sendo assim, ele considera o cascade.

Possui um pior desempenho, pois caso um objeto passado não esteja associado a sessão, é feito um **merge**, antes da remoção para inserir este objeto na sessão.

11.2.12. Query personalizada

Algumas vezes é necessário fazer consultas personalizadas, onde os métodos genéricos citados não abrangem.

Exemplo - Consultar pessoas com nome "Maria" ou nome "Joao" e que possuam carga horaria de 6h.



A principal função deste tópico foi mostrar que o `entityManager`, e entidade (`Pessoa.class`) é acessível nas classes que implementam o `BaseDAO` e algumas boas práticas para criação de Query. Na verdade o trecho acima no final das contas se refere a criação de um HQL.

11.3. Passando Parâmetros nos métodos do BaseDAO

Nos métodos do `BaseDAO` (`count`, `list`, etc...) pode-se passar 2 tipos de parâmetros: **Map** e **Restrictions**. `Restrictions` são mais aconselháveis, pois podem determinar o tipo (`like`, `equals`, `in`, etc...).

11.3.1. Map<String, Object>

String: nome do campo, **Object**: valor do campo.

Recuperar pessoas com o nome Maria:

```
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("nome", "Maria");
List<Pessoa> pessoas = pessoaDAO.list(parameters);
```

Podem ser inseridos vários objetos no `Map`, e pode-se passar a ordenação. O exemplo abaixo lista as pessoas de nome Maria e de carga horária 8h ordenados por **id**.

```
Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("nome", "Maria");
parameters.put("cargaHoraria", 8);
List<Pessoa> pessoas = pessoaDAO.list(parameters, "id");
```

A query gerada seria:

```
SELECT FROM Pessoa WHERE nome = 'Maria' and cargaHoraria = 8
```

É importante destacar que o **HashMap** não guarda a ordem de adição dos atributos, então a ordem dos campos após a cláusula **WHERE** pode não ser preservada. Para preservar a ordem pode-se usar **LinkedHashMap**.

11.3.2. Restrictions(List<Restriction>) e Restriction

`Restrictions` são mais eficientes que o `Map`, pois é possível dizer o tipo de Restrição.

Os tipos de restrição são:

Tipo de Restrição	Descrição
RestrictionType.EQUALS	igual a
RestrictionType.NOT_EQUALS	diferente de
RestrictionType.GREATER_THAN	maior que
RestrictionType.LESS_THAN	menor que
RestrictionType.GREATER_EQUALS_THAN	maior ou igual que
RestrictionType.LESS_EQUALS_THAN	menor ou igual que
RestrictionType.LIKE	que iniciam, terminam ou contenha
RestrictionType.NOT_LIKE	que não iniciam, não termine e não contenha
RestrictionType.IN	que estejam contidos em
RestrictionType.NOT_IN	que não estejam contidos em
RestrictionType.NULL	que seja null
RestrictionType.NOT_NULL	que não sejam null
RestrictionType.DATE_TABLE_FILTER	Tipo especial utilizado no LazyDataModelImpl

Para restrições do tipo **LIKE** ou **NOT_LIKE**, é possível definir, se é like no início, no fim ou ambos (padrão). Os possíveis tipos são:

Tipo de Restrição LIKE	Descrição
LikeType.BEGIN	que inicie com
LikeType.END	que termine com
LikeType.BOTH	que contenha (padrão quando nenhum é definido)

Exemplos:

Recuperar pessoas com o nome Maria, ordenados por **id**:

```
List< Restriction> restrictions = new ArrayList<Restriction>();
restrictions.add(new Restriction("nome", "Maria"));
pessoas = pessoaDAO.list(restrictions, "id");
```

Na versão 1.4 é possível utilizar a classe **Restrictions** para facilitar a criação de listas de restrições, o código acima mostrado seria o equivalente a:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "Maria");
pessoas = pessoaDAO.list(restrictions, "id");
```

Pode ser usado ainda em cadeia:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", "Maria").add("cargo", "Auxiliar");
pessoas = pessoaDAO.list(restrictions);
```

Recuperar pessoas que possuem a String *“Silva”* no nome:

```
Restrictions restrictions = new Restrictions();
restrictions.add("nome", RestrictionType.LIKE, "Silva");
pessoas = pessoaDAO.list(restrictions, "nome");
```

Recuperar pessoas que possuem carga horaria entre **6h** e **8h**:

```
Restrictions restrictions = new Restrictions();  
restrictions.add("cargaHoraria", RestrictionType.GREATER_EQUALS_THAN, 6);  
restrictions.add("cargaHoraria", RestrictionType.LESS_EQUALS_THAN, 8);  
pessoas = pessoaDAO.list(restrictions);
```

Existem muitas combinações possíveis quando se trabalha com Restrictions.

12. DataTable paginado no banco com o LazyDataModelImpl

Para criar um **LazyDataModel** (Primefaces) com paginação real no banco de dados, o xpert-framework disponibiliza a classe **LazyDataModelImpl** (com.xpert.faces.primefaces).

Esta classe recebe como parâmetro o DAO da entidade a ser consultada.

12.1. Utilização Básica

A utilização básica, basta passar a ordenação e o DAO.

Exemplo - Recuperar ofertas ordenadas por data de cadastro (da maior para menor):

```
LazyDataModel ofertas = new LazyDataModelImpl<Oferta>("dataCadastro DESC", ofertaDAO);
```

12.2. Adicionar restrições na consulta do LazyDataModel

Para consultas mais complexas, pode-se utilizar **Restrictions** para filtrar a consulta.

Exemplo - O método abaixo retorna as ofertas de um determinado site, que contenha uma determinada descrição, ordendos por descrição.

```
public LazyDataModel<Oferta> getOfertas(Site site, String descricao) {  
    Restrictions restrictions = new Restrictions();  
    restrictions.add("site", site);  
    restrictions.add("descricao", RestrictionType.LIKE, descricao);  
  
    return new LazyDataModelImpl<Oferta>("descricao", restrictions, ofertaDAO);  
}
```

12.3. Filtros da coluna

O **LazyDataModelImpl** faz a filtragem no banco a partir do campo definido no atributo "filterBy" definido no componente "column". Para isso ele utiliza o tipo de restrição *RestrictionType.DATA_TABLE_FILTER* seguindo a seguinte regra:

Tipo de Campo	Descrição
String	Utiliza consulta "like" (que contenha a String seja no início, meio ou fim).
Long, Integer, BigDecimal	Utiliza consulta "igual a"
Boolean	Utiliza consulta "igual a". Para evitar que tenha que ser digitado "true" ou "false" pode ser utilizado o filterOptions com o "booleanSelectItemsEmptyOptions". Exemplo: filterOptions="#{booleanSelectItemsEmptyOption}".
Date e Calendar	Tenta consultar pela data convertida em String, a data deve seguir o padrão java "SimpleDateFormat.MEDIUM" do Locale Atual. E caso sejam passados 2 valores separados por hífen "-" é feita uma consulta de intervalos e as 2 datas. Exemplo(pt_BR): "01/01/13 - 01/02/13" traria os registros entre essas 2 datas. O uso do componente DateFilter facilita essa filtragem.

13. Unicidade dos campos com UniqueField

Para verificar a unicidade dos campos, pode-se utilizar o objeto **UniqueField**. Com ele é possível definir um campo único, ou combinações de campos.

13.1. Definindo campos únicos em um BO genérico

Ao estender a classe **AbstractBusinessObject** (com.xpert.core.crud) é obrigatório implementar o método **getUniqueFields()**.

Exemplo - BO sem validação de campos únicos:

```
@Override
public List<UniqueField> getUniqueFields() {
    return null;
}
```

Exemplo - BO onde a entidade não pode ter a descrição repetida:

```
@Override
public List<UniqueField> getUniqueFields() {
    return new ArrayList<UniqueField>() {
        {
            add(new UniqueField("descricao"));
        }
    };
}
```

A partir da versão 1.4 foi disponibilizada a class **UniqueFields**, para facilitar criação de listas de campos únicos, o código mostrado acima seria o equivalente a:

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("descricao");
}
```

Pode ainda ser usado em cadeia (no exemplo o objeto não poderia possuir a mesma descrição, nem o mesmo código):

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("descricao").add("codigo");
}
```

13.2. Customização de mensagem no UniqueField

Arquivo de Internacionalização:

```
pais.negocio.descricaoUnica=Pais já cadastrado com essa descrição
```

Utilização:

```
@Override
public List<UniqueField> getUniqueFields() {
    List<UniqueField> uniqueFields = new ArrayList<UniqueField>();
    uniqueFields.add(new UniqueField("descricao").setMessage("Custom Message"));
    return uniqueFields;
}
```

O código acima também pode ser utilizado assim:

```
@Override
public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add(new UniqueField("descricao"), "Custom Message");
}
```

13.3. Validação fora de um BO Genérico

Exemplo - Não pode existir pessoa com CPF duplicado:

```
import com.xpert.core.exception.UniqueFieldException;
import com.xpert.core.validation.UniqueFieldsValidation;
import com.xpert.faces.utils.FacesMessageUtils;

public List<UniqueField> getUniqueFields() {
    return new UniqueFields().add("cpf");
}

public void save(Pessoa pessoa){
    try {
        UniqueFieldsValidation.validateUniqueFields(getUniqueFields(), pessoa, pessoaDAO);
    } catch (UniqueFieldException ex) {
        FacesMessageUtils.error(ex);
    }
}
```

com.xpert.core.validation.UniqueFieldsValidation
xpert-framework
Classe para realizar validação dos campos únicos.

validateUniqueFields()
xpert-framework

recebe como parâmetro, uma lista de campos únicos, o objeto a ser validado, e o dao. Este método lança a exceção **UniqueFieldException**

UniqueFieldException é um **BusinessException**, sendo assim, ao ser capturado ele pode ser exibido formatado como **FacesMessageUtils**, como mostrado no exemplo acima.

14. Criação de relatórios com o FacesJasper

Para criar relatórios *jasperreports* pode-se utilizar a classe FacesJasper (com.xpert.faces.utils).

Exemplo de criação de um relatório:

```
FacesJasper.createJasperReport(dataSource, parameters, "WEB-INF/reports/report.jasper", "report.pdf");
```

A geração de relatório gera um download de arquivo, então ela pode ser combinada com o componente **download**, isso causa uma melhor experiência ao usuário, pois este componente bloqueia a tela enquanto o download é gerado:

Exemplo:

```
<p:commandButton value="Create Report" action="#{reportMB.generateReport}" ajax="false">
  <x:download/>
</p:commandButton>
```

Para componentes ajax, com **commandLink** e **commandButton** do primefaces, é necessário especificar **ajax="false"**, visto que um download não pode ser ajax.

15. Geração de Código (CRUD) com o Xpert-Maker

15.1. Introdução

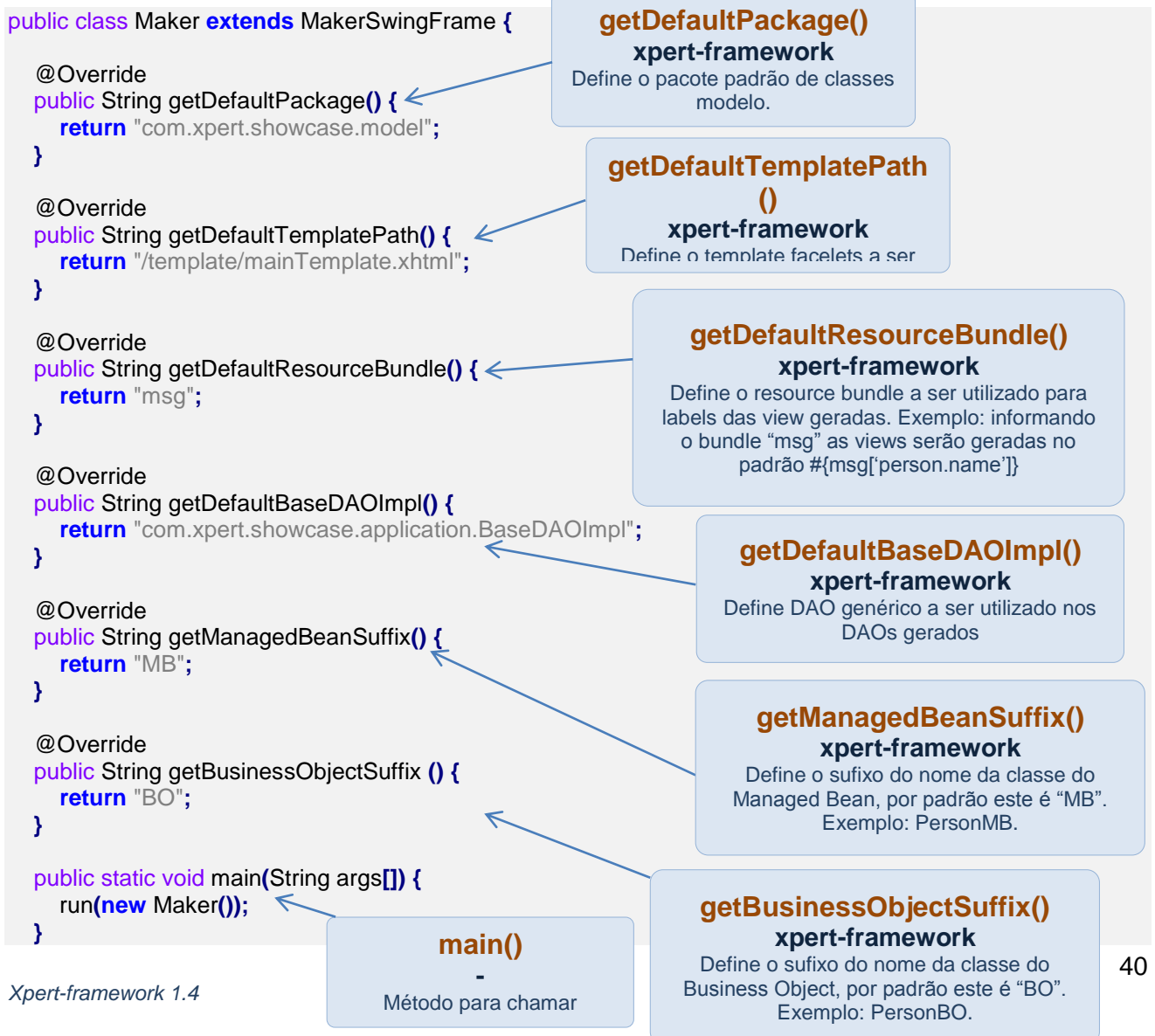
A geração de código pode ser feita de 2 maneiras a primeira delas é via Swing (disponibilizado a partir da versão 1.3), onde o código é gerado diretamente no projeto do desenvolvedor e a segunda é via Componente JSF onde é disponibilizada uma interface para que o código seja gerado, por ser web as classes não são salvas diretamente na máquina do desenvolvedor, em vez disso é exibido o código na tela e disponibilizado o download do código-fonte compactado.

15.2. Utilização Básica via componente Swing

A partir da versão 1.3 do xpert-framework foi criado uma aplicação swing para geração de código, a vantagem é que o código é gerado diretamente no projeto do desenvolvedor.

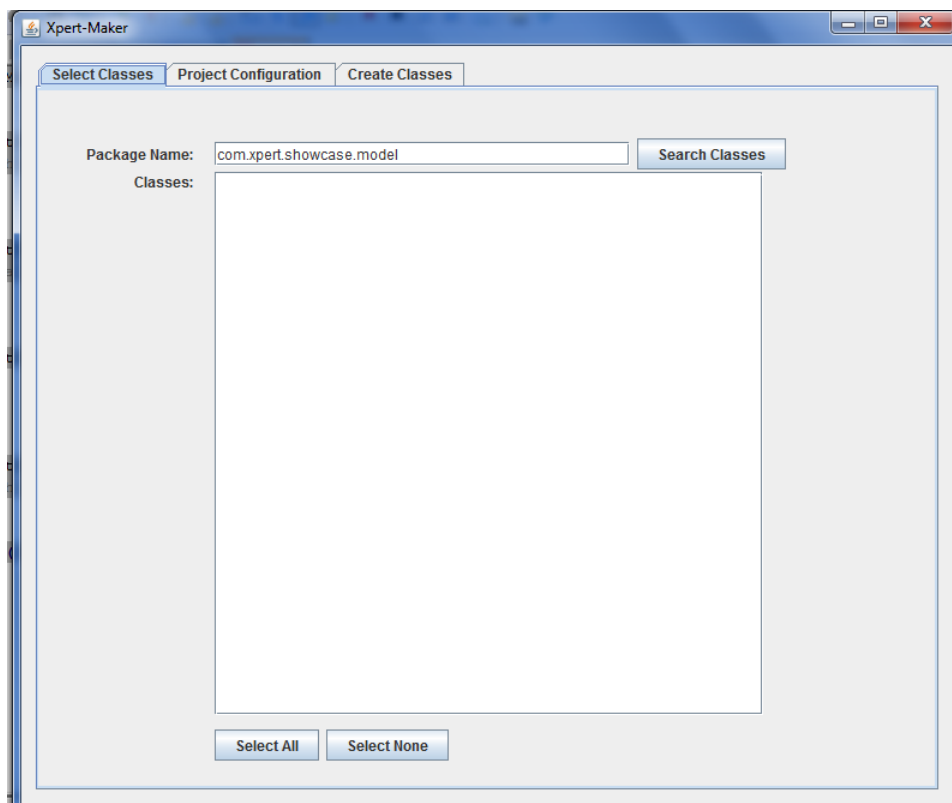
Primeiramente deve ser criada uma classe geradora no projeto que possui as classes modelo. Essa classe deve ser filha de **MakerSwingFrame** (pacote com.xpert.maker) sendo que ela é uma classe abstrata onde alguns métodos devem ser sobrescritos.

Abaixa segue o exemplo de uma classe:



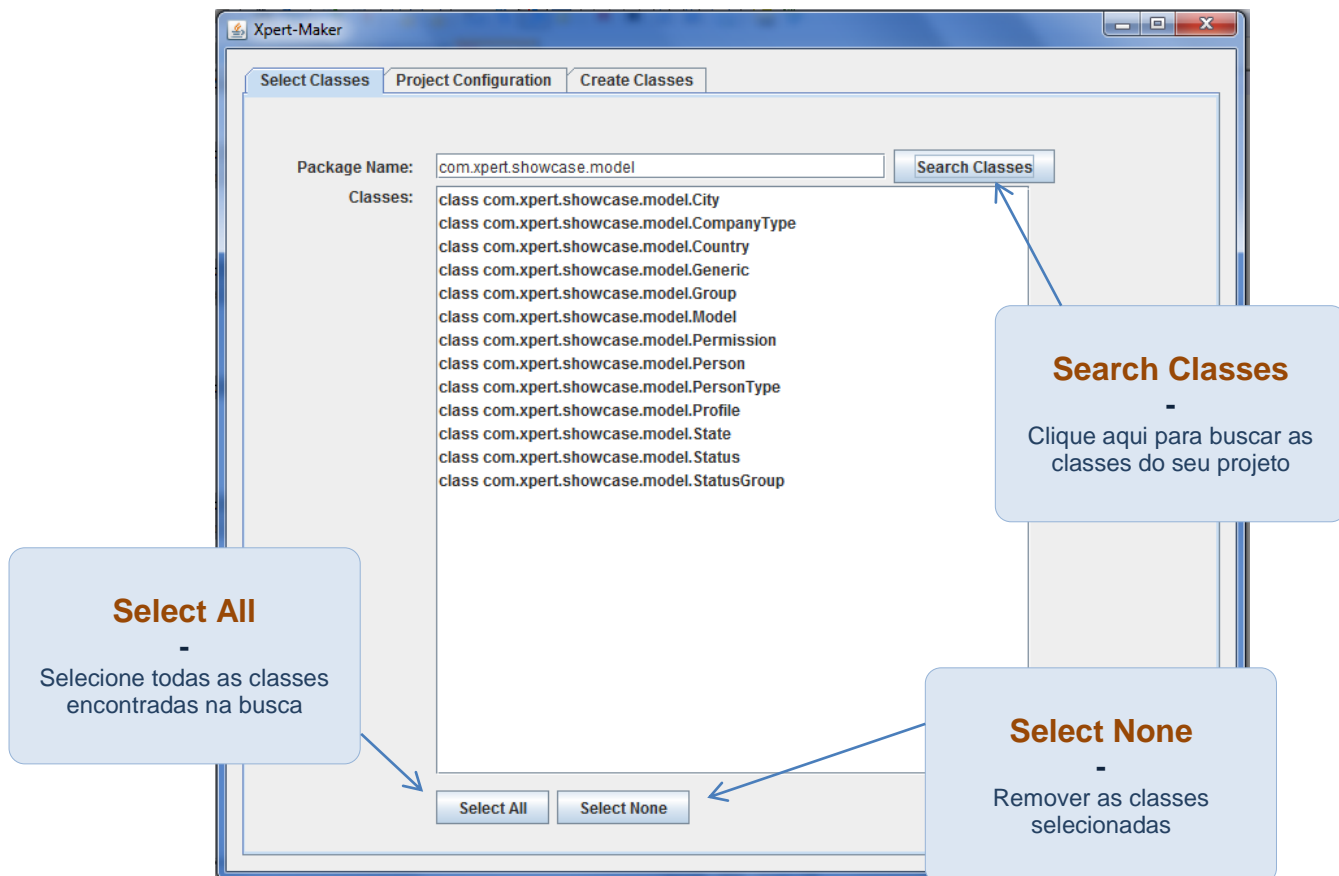
}

Após a classe acima ser executada o resultado é o seguinte:



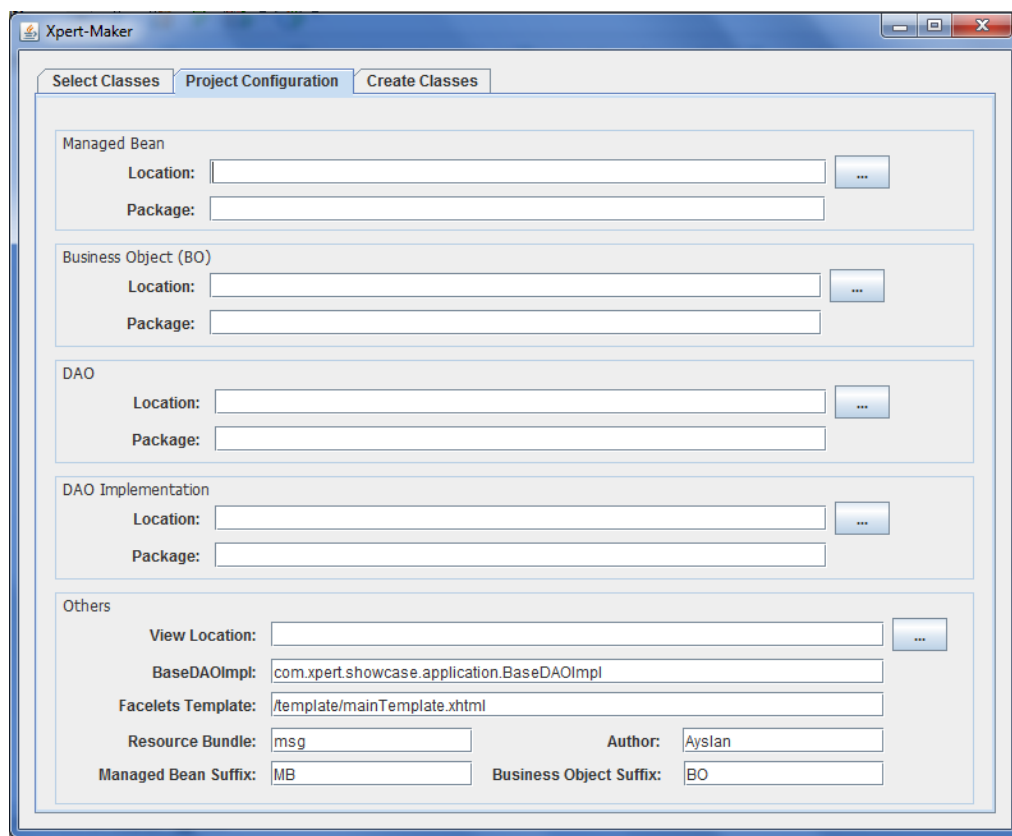
O campo Package name é utilizado para listar as classe do pacote informado, por padrão ele vem preenchido com o valor informado no método **getDefaultPackage()**.

Para buscar as classes clique no botão **“Search Classes”**, o resultado será o seguinte:



Selecione as classes que você deseja gerar, para selecionar todos clique em **“Select All”** e para remover a seleção clique em **“Select None”**.

Vá para a próxima aba **“Project Configuration”**:



Note que o painel “**Others**” dessa aba vem preenchido com os métodos que foram sobrescritos na classe.

Nessa aba você deve informar o caminho das suas classes, por exemplo informar onde vai ficar o Managed Bean, DAO, DAOImpl, BO e onde suas views serão geradas (campo “**Location**”).

Ainda nessa aba também informe o padrão de pacotes para suas classes (campo “**Package**”).

Abaixo como esta aba ficaria preenchida:

Selecionar Pasta
Clique aqui para selecionar a pasta da classe a ser gerada

Xpert-Maker

Select Classes | **Project Configuration** | Create Classes

Managed Bean

Location: \Desenvolvimento\xpert-projects\xpert-showcase\src\main\java\com\xpert\showcase\mb ...

Package: com.xpert.showcase.mb

Business Object (BO)

Location: \Desenvolvimento\xpert-projects\xpert-showcase\src\main\java\com\xpert\showcase\bo ...

Package: com.xpert.showcase.bo

DAO

Location: \Desenvolvimento\xpert-projects\xpert-showcase\src\main\java\com\xpert\showcase\dao ...

Package: com.xpert.showcase.dao

DAO Implementation

Location: \Desenvolvimento\xpert-projects\xpert-showcase\src\main\java\com\xpert\showcase\dao\impl ...

Package: com.xpert.showcase.dao.impl

Others

View Location: ...

BaseDAOImpl: com.xpert.showcase.application.BaseDAOImpl

Facelets Template: /template/mainTemplate.xhtml

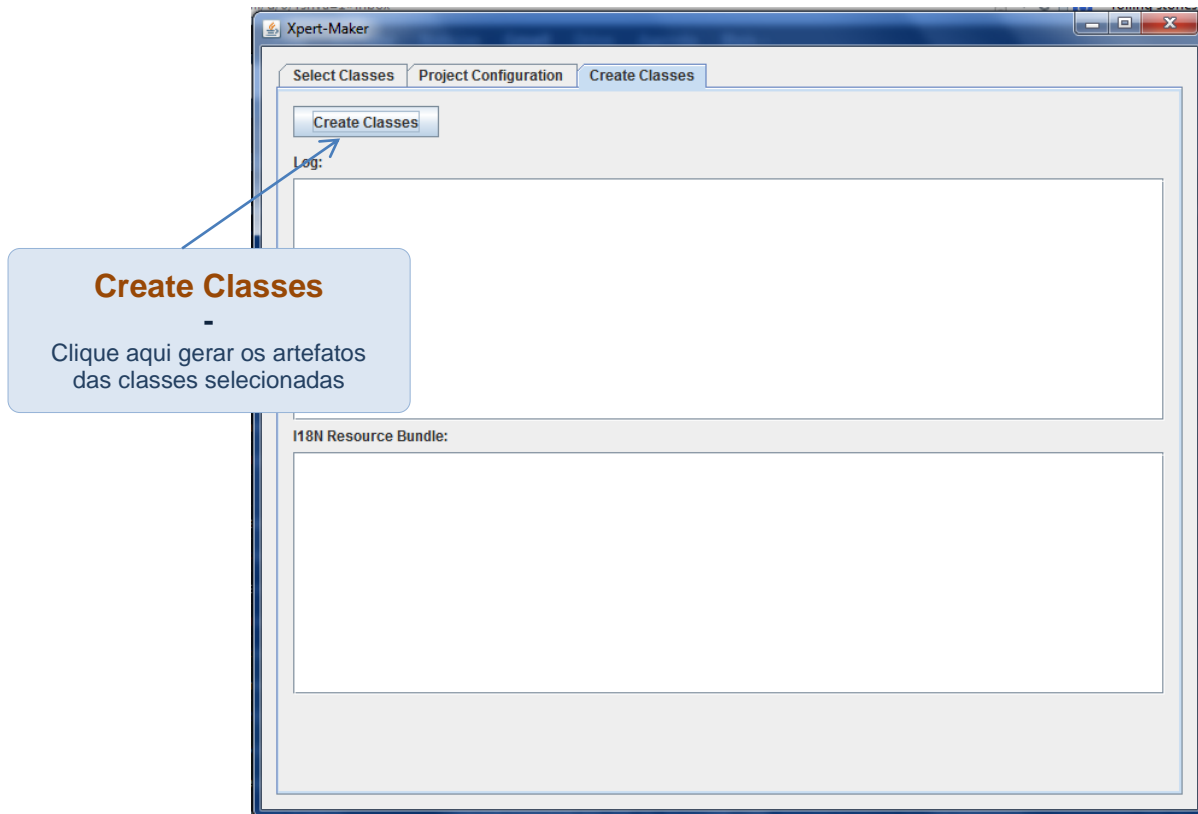
Resource Bundle: msg

Managed Bean Suffix: MB

Author: Ayslan

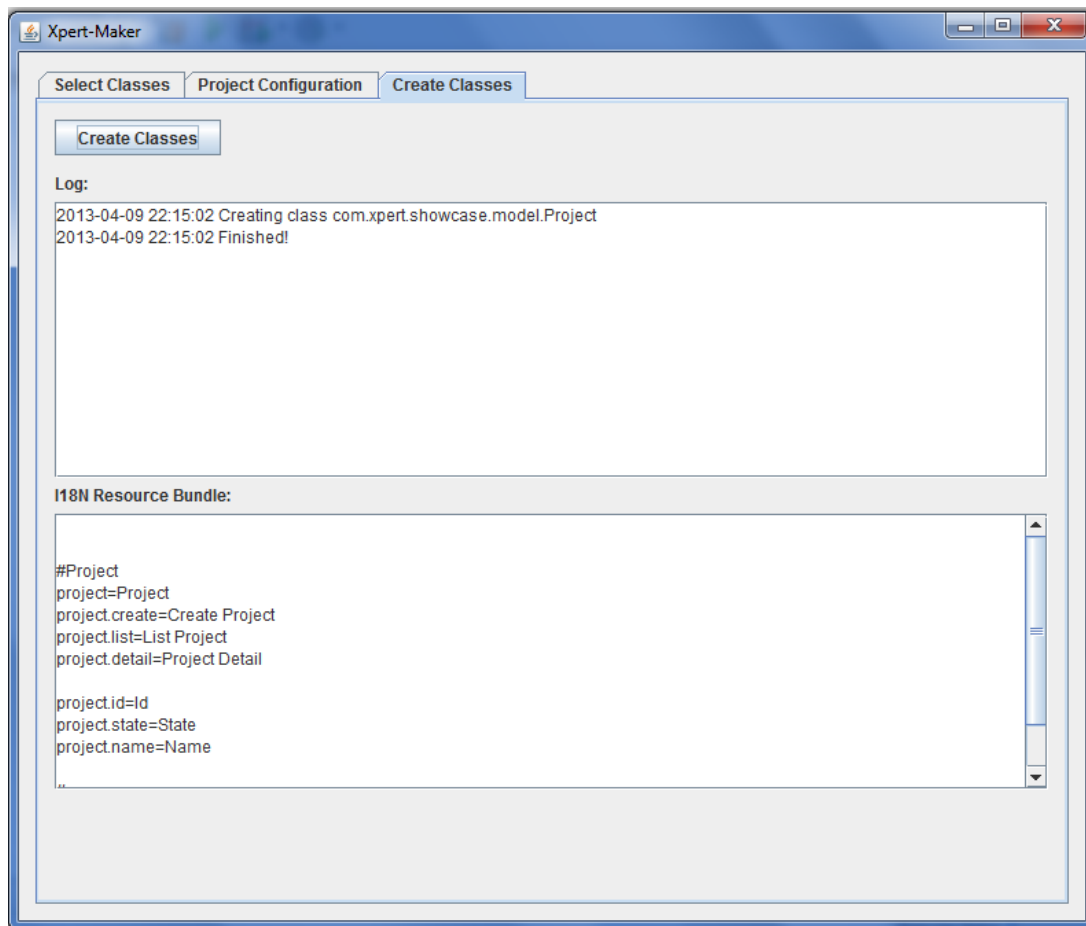
Business Object Suffix: BO

O passo seguinte é gerar as classes, vá para a aba “**Create Classes**”:



Clique em “**Create Classes**” e as classes serão escritas no seu projeto. O log de geração pode ser acompanhado no campo “**Log**”. As mensagens de internacionalização podem ser pegadas no campo “**I18N Resource Bundle**”.

Abaixo o resultado de uma geração com sucesso:



15.3. Utilização Básica via componente JSF

A geração de código é feita baseada em uma entidade já existente.

Para utilizar o componente xpert-maker é necessário criar um EntityManagerFactory e defini-lo no arquivo xpert-config.xml para que o framework encontre o entity manager da aplicação.

O componente de geração de código chama-se **maker** possui a seguinte estrutura:

Adicionar o namespace no xhtml:

```
xmlns:xc="http://java.sun.com/jsf/composite/xpert/components"
```

Chamada do componente:

```
<xc:maker managedBean="com.xpert.showcase.mb.crud"
  businessObject="com.xpert.showcase.bo"
  dao="com.xpert.showcase.dao"
  daoImpl="com.xpert.showcase.dao.impl"
  baseDAO="com.xpert.showcase.dao.BaseDAOImpl"
  resourceBundle="msg"
  template="/template/mainTemplate.xhtml"
  viewPath="/view"
  author="Author" />
```

Os atributos do componente maker são:

Atributo	Descrição
managedBean	Nome padrão dos pacotes dos ManagedBeans gerados.
businessObject	Nome padrão dos pacotes dos BusinessObjects (BOs) gerados.
dao	Nome padrão dos pacotes dos DAOs (interface) gerados.
daoImpl	Nome padrão dos pacotes da implementação dos DAOs gerados.
baseDAO	Nome completo da classe padrão a ser utilizada como implementação do BaseDAO do xpert-framework.
resourceBundle	Nome do Resource Bundle a ser usado para acessar o arquivo de internacionalização.
template	Caminho do template a ser utilizada nos XHTML gerados.
viewPath	Caminho padrão das páginas geradas
author	Autoria da geração de código

Após inserido na tela o maker é renderizado da seguinte maneira:

A primeira aba (**Configuration**) é usada para configurações do projeto:

1 - Configuration
-
Aba para definição de configurações do projeto.

Package Configuration
-
Configuração de nomes dos pacotes do projeto.

1 - Configuration
2 - Create Classes

Package Configuration

Insert the packages name

Managed Bean:

DAO:

DAO Impl:

Business Object:

Others

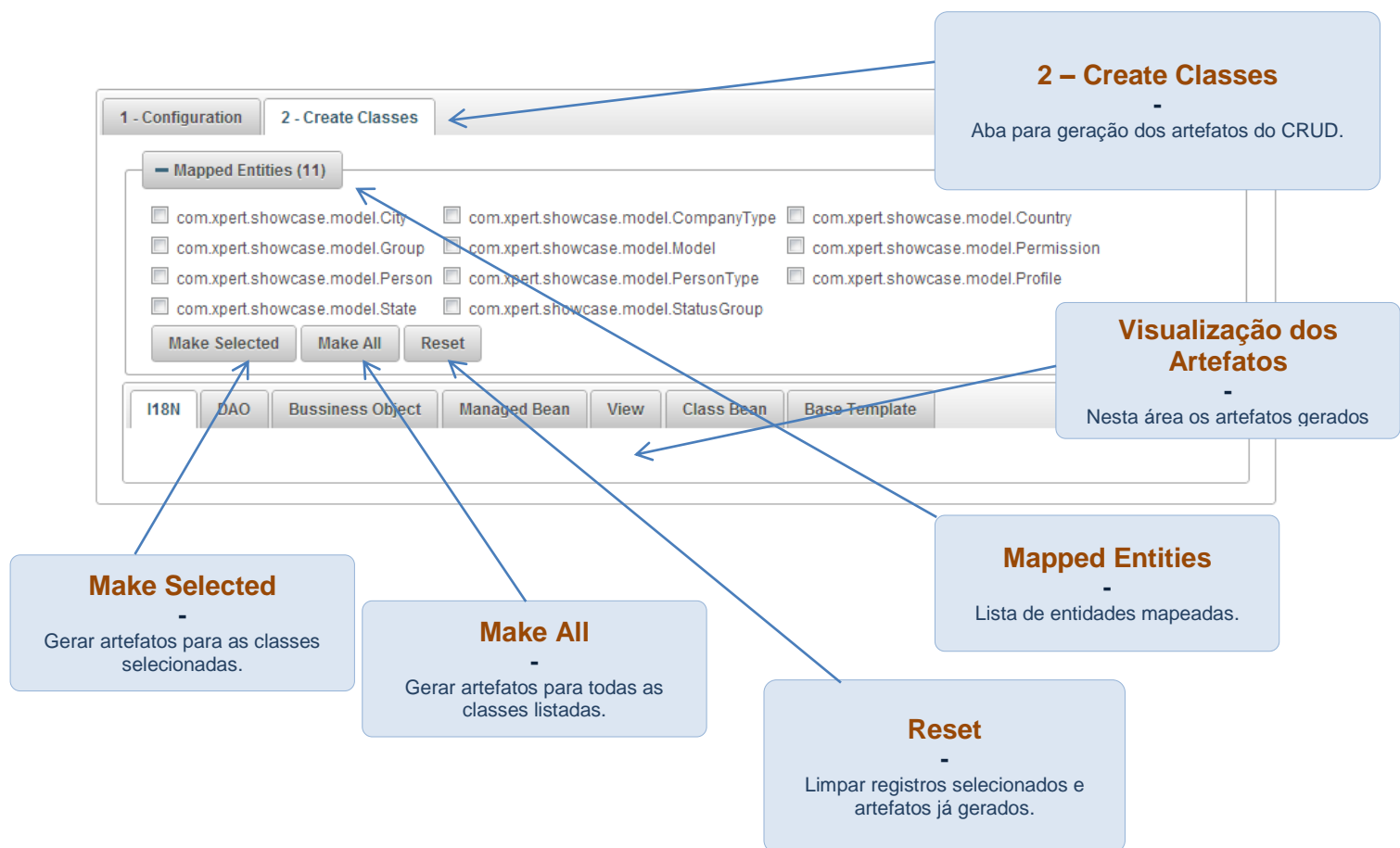
BaseDAOImpl Class: The BaseDAO class of your Project (Must extend com.xpert.persistence.dao.BaseDAOImpl)

Faces Resource Bundle:

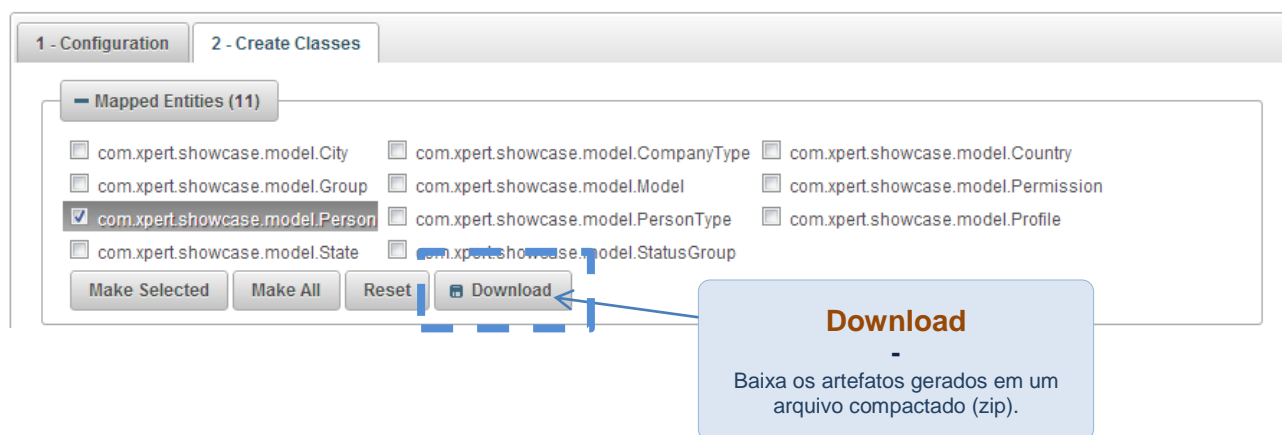
Template (XHTML):

Author:

Com a configuração do projeto realizada, basta ir para a segunda aba (**Create Classes**):

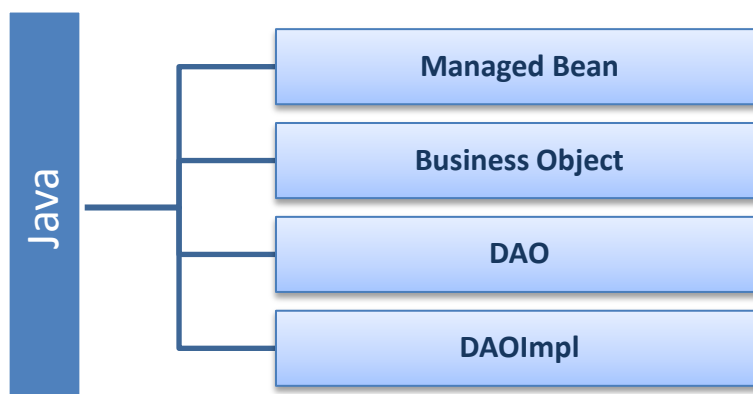


Clicando em **Make Selected** ou em **Make All** será exibido o resultado da geração de código, os artefatos podem ser baixados através do botão **Download**:

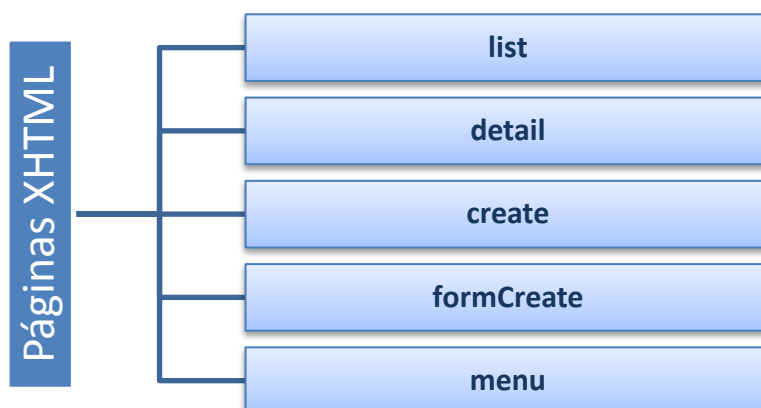


15.4. Estrutura de um CRUD gerado

15.4.1. Artefatos criados para cada Entidade



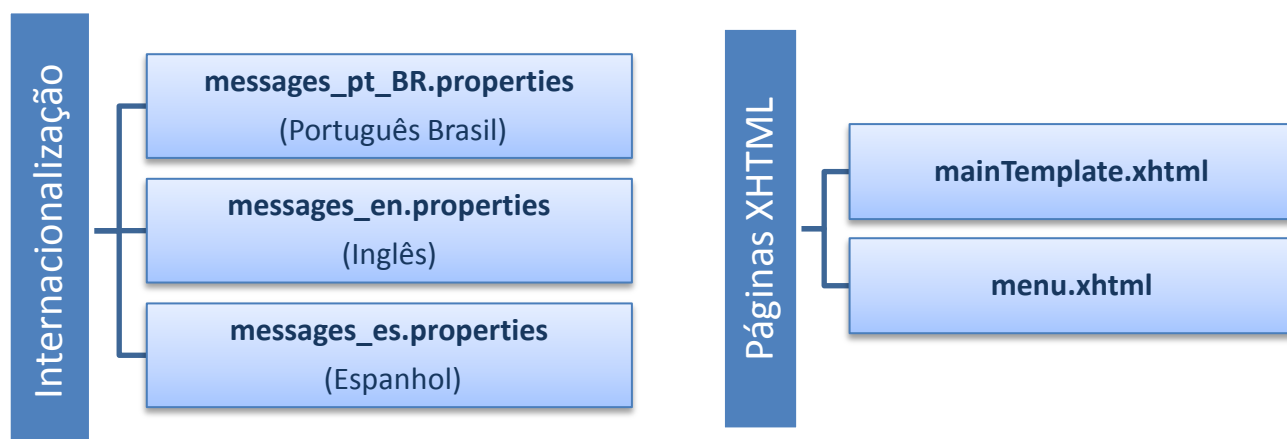
Classe	Descrição
Managed Bean	Managed Bean do JSF seguindo a especificação, as classes geradas recebem a anotação @ManagedBean e o escopo @ViewScoped . O padrão da classe pode ser visto na sessão Estrutura de um ManagedBean .
Business Object	Classe de regra de negócio. Esta classe é um session bean da especificação do EJB, recebe a anotação @Stateless . O padrão da classe pode ser vista em Estrutura de um Business Object .
DAO	Interface que segue o padrão Data Access Object .
DAO Impl	Implementação do DAO.



XHTML	Descrição
list	Página de listagem da entidade, nela é possível: detalhar, excluir, auditar exclusões e detalhar. Esta página recebe o include da view detail e do menu .
detail	Detalhamento da entidade, nele é possível ainda auditar da entidade.
create	Página de criação da entidade. Esta página recebe include do formCreate e do menu .
formCreate	Form para criação da entidade. Esta página é separada do create pois assim fica mais dinâmica e pode ser utilizada em forma da includes em outras páginas.
menu	Possui os links de acesso às páginas de create e list

15.4.2. Artefatos únicos

Esses são artefatos onde geralmente só se existe um por projeto. Eles só são gerados a partir do componente JSF não sendo possível a geração através do componente Swing.

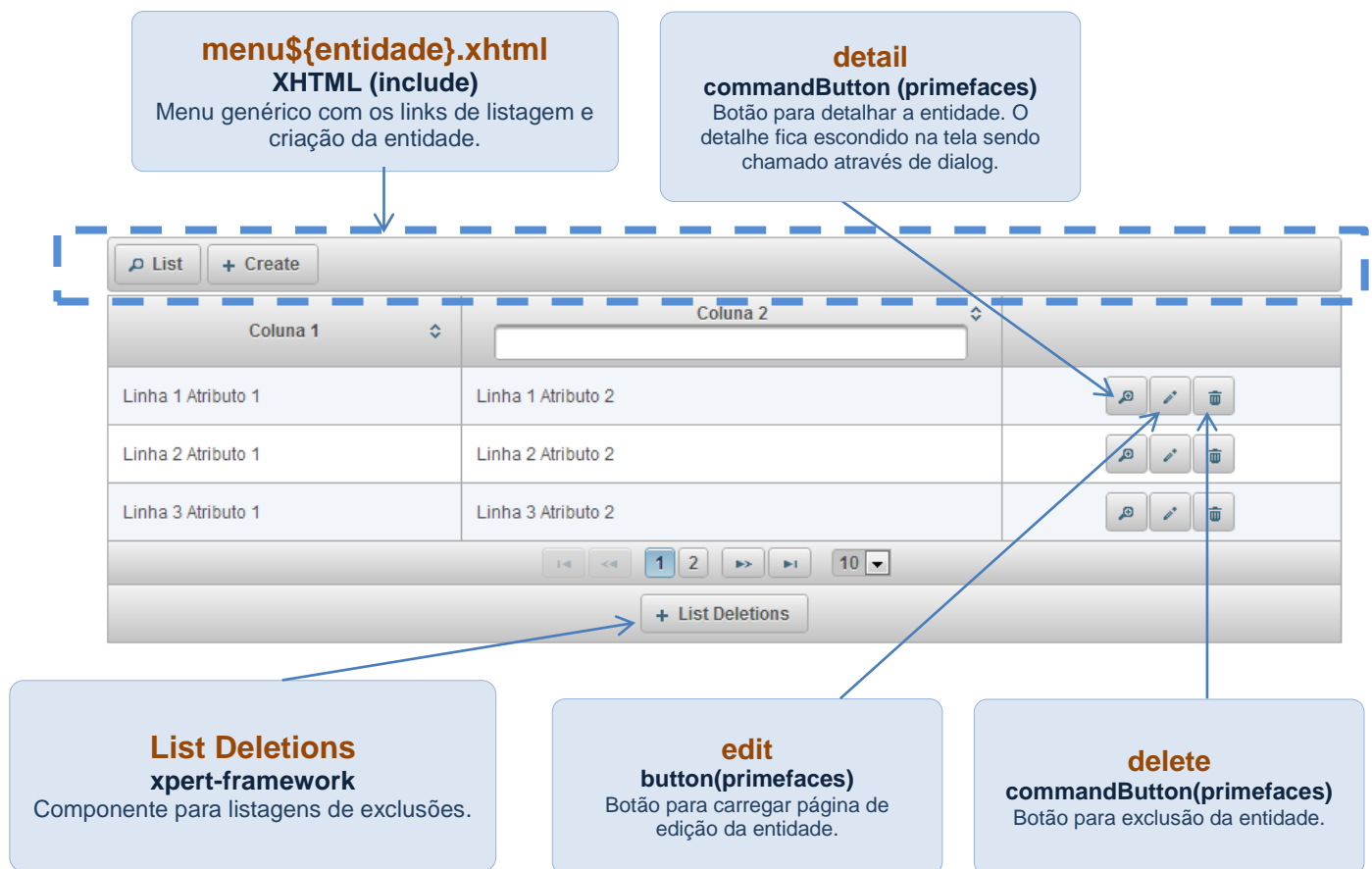


Artefato	Descrição
messages	Arquivo de internacionalização.
mainTemplate.xhtml	Template genérico para o projeto.
menu.xhtml	Possui os links de acessos aos CRUDs gerados.

15.4.3. Padrão da view para listagem de Registros – list{entidade}.xhtml

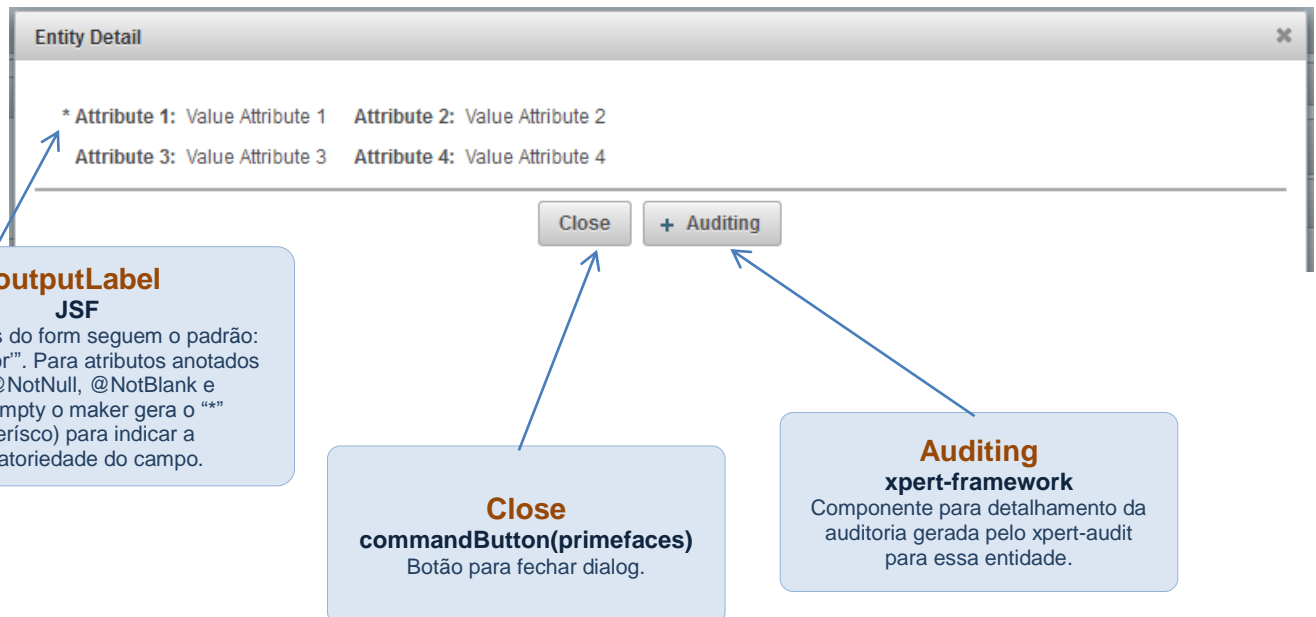
A view de listagem consiste em um **dataTable** (LazyDataModel paginado no banco), sendo que para cada registro se tem a opção de **detalhar**, **editar** e **excluir**. Essa tabela ainda a opção de listar as exclusões (que foram capturadas a partir do xpert-audit).

As colunas são geradas dinamicamente a partir dos atributos da entidade, ou seja, para cada atributo existe uma coluna (com exceção os tipos de listas (List, Collection, etc...)).



15.4.4. Padrão da view para detalhamento do registro – detail{entidade}.xhtml

A view de detalhamento é usada na forma de include (sendo exibida dentro de um dialog) para a listagem. Basicamente ele lista os atributos da classe em forma de texto. O detalhamento ainda gera o componente “Auditing” para detalhe da auditoria da entidade.



15.4.5. Padrão da view para criação e edição do registro – create{entidade}.xhtml

O XHTML gerado para a criação de views consiste basicamente em includes do **menu** e do **formCreate**. Os campos em si estão concentrados no xhtml **formCreate{entidade}.xhtml**. Essa separação possibilita futuras reutilizações na forma de include para o form.

outputLabel
JSF

Os campos do form seguem o padrão: "label: 'valor'". Para atributos anotados com @NotNull, @NotBlank e @NotEmpty o maker gera o "*" (asterisco) para indicar a obrigatoriedade do campo.

menu\${entidade}.xhtml
XHTML (include)

Menu genérico com os links de listagem e criação da entidade.

formCreate\${entidade}.xhtml
XHTML (include)

Form com os campos da entidade.

The diagram illustrates a web form layout with several components and annotations:

- Navigation Bar:** A horizontal bar at the top containing a "List" button with a magnifying glass icon and a "+ Create" button.
- Form Fields:** Below the navigation bar, there are three required fields: "* Code:" (text input), "* Name:" (text input), and "* Group:" (dropdown menu with "Select" as the current value).
- Permissions:** A row of checkboxes labeled "Permissions:" with options: CREATE, DELETE, DELETE CASCADE, SEARCH, and UPDATE.
- Legend:** A line labeled "(*) Campos Obrigatórios" (Required Fields).
- Save Button:** A "Save" button located at the bottom right of the form area.

Annotations with arrows point to specific parts of the form:

- A blue arrow points from the **outputLabel JSF** box to the asterisk (*) next to the "Code:" label.
- A blue arrow points from the **menu\${entidade}.xhtml XHTML (include)** box to the "+ Create" button.
- A red arrow points from the **formCreate\${entidade}.xhtml XHTML (include)** box to the "Name:" text input field.
- A blue arrow points from the **Save commandButton (primefaces)** box to the "Save" button.

15.5. Atributos mapeados e seus respectivos componentes na View

Cada tipo de atributo possui um componente correspondente gerado na tela através do maker, por exemplo uma **String** corresponde a um `inputText` (`p:inputText`). Abaixo segue a lista completa por tipo de atributo.

15.5.1.1. Atributos java

Tipo de Atributo	Componente Renderizado	Tipo de Componente
String	<code>p:inputText</code>	Primefaces
boolean	<code>h:selectBooleanCheckbox</code>	JSF
BigDecimal e Double	<code>xc:inputNumber</code>	Xpert-framework
Integer e Long	<code>p:inputMask</code> (mask = 9?999999999)	Primefaces
Date e Calendar	<code>p:calendar</code>	Primefaces

15.5.1.2. Atributos de relacionamentos JPA

Tipo de Atributo	Componente Renderizado	Tipo de Componente
@ManyToOne	<code>h:selectOneMenu</code>	JSF
@OneToMany e @ManyToMany	<code>h:selectManyCheckbox</code>	JSF

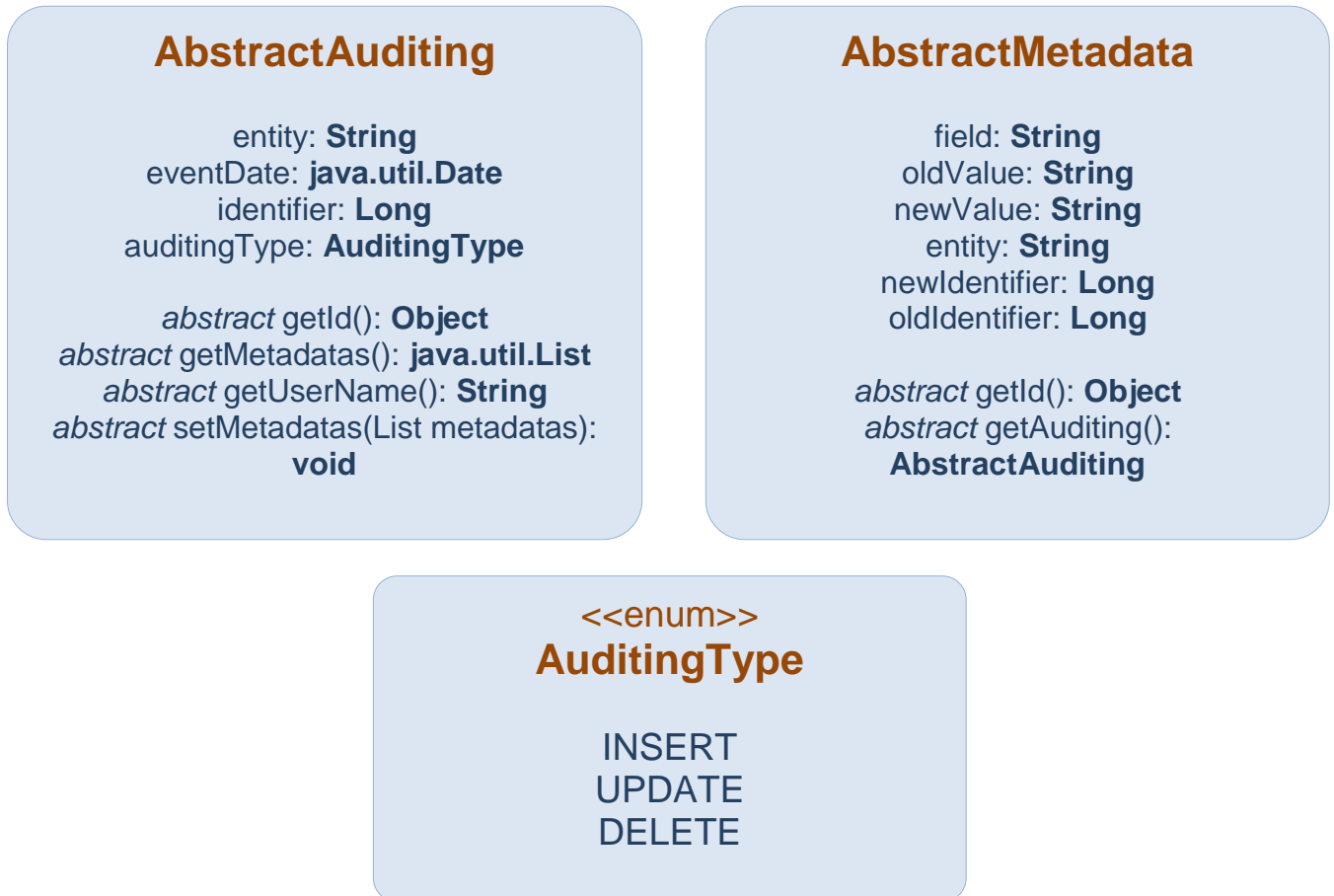
Observações:

- Para relacionamentos que estejam LAZY, o xpert-maker insere o componente **x:initializer**.
- Para pegar o objeto por completo os componentes **h:selectOneMenu** e **h:selectManyCheckbox** exigem um conversor, para esses casos o xpert-maker insere o conversor genérico **entityConverter**.
- Para campos boolean na exibição na listagem e no detalhamento é inserido o conversor **yesNoConverter** para exibição de Sim (true) e Não (false) na view.

16. Auditando as entidades com o Xpert-Audit

16.1. Introdução

O objetivo desse módulo do framework é auditar as entidades na inserção, atualização e exclusão. As classes de auditoria seguem o seguinte modelo:



16.2. AbstractAuditing - Classe que representa uma auditoria feita

16.2.1. Introdução

Esta classe representa uma auditoria feita para uma entidade. Caso um evento que gera auditoria ocorra, um objeto deste tipo será criado, por exemplo, cada inserção gera um *Auditing*, assim como atualização e exclusão. Por exemplo, ao se salvar um objeto da classe City seria criado um registro da seguinte maneira:

Detalhamento da Auditoria			
	Data	Tipo	Usuário
+	18/03/2013 23:44:21	Inclusão	
<div>1 10</div>			

16.2.2. Atributos

Atributo/Método	Tipo	Descrição
entity	String	Representa a entidade atual a ser auditada. Caso a anotação @Table esteja presente, o valor dela é usado, caso a anotação @Entity esteja com valor definido este é usado, caso contrário usa-se o nome simples da classe.
eventDate	java.util.Date	Data da auditoria.
identifier	Long	Identificador do objeto auditado.
auditingType	AuditingType	Enum que representa o tipo de auditoria (insert, update ou delete)
getId()	Object	Retorna o id do registro atual de auditoria
getMetadatas()	java.util.List	Retorna uma lista de metadados da auditoria atual
getUserName()	String	Método que retorna o usuário da auditoria, este método é usado no componente de auditoria.
setMetadatas()	void	Seta uma lista de metadados da auditoria atual.

16.2.3. Configuração

A implementação dessa classe deve ser declarada no arquivo xpert-config (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag **auditing-impl**.

16.2.4. Exemplo de uma implementação

O exemplo a seguir mostra o básico da classe, sendo adicionado apenas o objeto user, que seria simulado o usuário logado de uma aplicação, mais atributos também podem ser adicionados.


```

@Entity
public class Auditing extends AbstractAuditing implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    private Person user;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "auditing")
    private List<Metadata> metadatas;

    @Override
    public Long getId() {
        return id;
    }
    public Person getUser() {
        return user;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public void setUser(Person user) {
        this.user = user;
    }
    @Override
    public List getMetadatas() {
        return metadatas;
    }
    @Override
    public void setMetadatas(List metadatas) {
        this.metadatas = metadatas;
    }
    @Override
    public String getUsername() {
        if(user != null){
            return user.getName();
        }
        return "";
    }
}

```

16.3. AbstractMetadata - Classe que representa os metadados

16.3.1. Introdução

Quando um registro de auditoria é criado ele pode está vinculado a metadados, que são basicamente os valores dos campos naquele momento. Por exemplo a classe ao se salvar um objeto da Classe City será salvo os metadados da seguinte maneira:

Detalhamento da Auditoria			
	Data	Tipo	Usuário
-	18/03/2013 23:44:21	Inclusão	
	Campo	Valor Anterior	Valor Novo
1	name		Guarulhos
2	state		SÃO PAULO
3	creation		

1 10

16.3.2. Atributos

Atributo/Método	Tipo	Descrição
field	String	Representa o campo que foi auditado.
oldValue	String	Representa o valor original do registro auditado, para inserção esse valor é vazio.
newValue	String	Representa o valor novo do registro auditado.
entity	String	Nome da classe do tipo auditado quando este se refere a um objeto complexo.
newIdentifier	Long	Novo Identificador da entidade do campo auditado quando este é um objeto complexo.
oldIdentifier	Long	Identificador original da entidade do campo auditado quando este é um objeto complexo.
getId()	Object	Retorna o id do registro atual do metadado.
getAuditing()	AbstractAuditing	Retorna o registro de auditoria para o metadado.

16.3.3. Configuração

A implementação dessa classe deve ser declarada no arquivo `xpert-config` (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag **metadata-impl**.

16.3.4. Exemplo de uma implementação

O exemplo a seguir mostra o básico da classe onde o atributo **auditing** é do tipo **Auditing**, mostrado na seção anterior.

```

@Entity
public class Metadata extends AbstractMetadata{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    private Auditing auditing;

    @Override
    public Long getId() {
        return id;
    }

    @Override
    public AbstractAuditing getAuditing() {
        return auditing;
    }

    @Override
    public void setAuditing(AbstractAuditing auditing) {
        this.auditing = (Auditing) auditing;
    }
}

```

16.4. Listener para a auditar um objeto

16.4.1. Introdução

Em uma aplicação real uma auditoria deve-se indicar o usuário que realizou a operação, o xpert-framework deixa de maneira genérica essa parte. Para isso é disponibilizado a classe `AbstractAuditingListener`, através dela o projeto pode ter uma implementação própria e nela fazer sua lógica.

16.4.2. Configuração

A implementação dessa classe deve ser declarada no arquivo `xpert-config` (mais detalhes sobre este arquivo na seção [Configurando o xpert-config.xml](#)) através da tag `auditing-listener`.

16.4.3. Exemplo de uma implementação

O exemplo a seguir se baseia na implementação do tipo **Auditing**, mostrado na seção anterior. O método `SessionUtils.getUser()` é apenas um exemplo, nesse trecho deve ser feita a lógica para setar o usuário.

```

public class AuditingListenerImpl implements AbstractAuditingListener {

    @Override
    public void onSave(AbstractAuditing abstractAuditing) {
        Auditing auditing = (Auditing)abstractAuditing;
        //example, set here the current user
        auditing.setUser(SessionUtils.getUser());
    }
}

```

16.5. Auditando uma entidade

Para auditar uma entidade deve-se chamar a classe Audit (com.xpert.audit). Esta classe possui os métodos para auditar a inserção, atualização e exclusão da entidade.

16.5.1. Auditar um Insert

Deve ser chamado após o método de salvar

```
public void save(T object) {  
    getEntityManager().persist(object);  
    new Audit(getEntityManager()).insert(object);  
}
```

16.5.2. Auditar um Update

Deve ser chamado antes de o objeto ser persistido. O exemplo abaixo mostra como usar dentro de um método que utiliza o merge do JPA, já verificando se é um objeto novo ou um a ser atualizado:

```
public T merge(T object) {  
    boolean persisted = object.getId() != null;  
    if (persisted == true) {  
        new Audit(getEntityManager()).update(object);  
    }  
    object = (T) getEntityManager().merge(object);  
    if (persisted == false) {  
        new Audit(getEntityManager()).insert(object);  
    }  
    return object;  
}
```

16.5.3. Auditar um Delete

Deve ser chamado antes de se deletar a entidade. O exemplo abaixo utiliza o método remove do EntityManager:

```
public void remove(Object object){  
    new Audit(getEntityManager()).delete(object);  
    getEntityManager().remove(object);  
}
```

16.5.4. BaseDAOImpl do xpert-framework e sua auditoria

O DAO BaseDAOImpl(com.xpert.persistence.dao) do xpert-framework por padrão usa auditoria ao salvar ou deletar o objeto e o pode ser chamado a partir dos seguintes métodos:

```
public void save(T object);  
public T merge(T object);  
public void saveOrUpdate(T object);  
public void update(T object);
```

```
public void delete(Object id) throws DeleteException;
public void remove(Object object) throws DeleteException;
```

Para indicar se a entidade deve ou não ser auditada pode se chamar os métodos (o parâmetro *audit* indica se deve ser auditado):

```
public void save(T object, boolean audit);
public void saveOrUpdate(T object, boolean audit);
public void update(T object, boolean audit);
public void delete(Object id, boolean audit) throws DeleteException;
public void remove(Object object, boolean audit) throws DeleteException;
public T merge(T object, boolean audit);
```

16.6. Exibindo a auditoria de uma determinada entidade

O xpert-framework disponibiliza o componente “audit” e “auditDelete” para que a auditoria seja visualizada na tela através de componentes JSF e Primefaces.

Para realizar a consulta da auditoria o xpert-framework precisa que seja definido o EntityManagerFactory e que seja devidamente configurado, isso pode ser visto na sessão [Configurando o EntityManagerFactory](#).

Ambos os componentes estão acessíveis através do seguinte namespace:

```
xmlns:xc="http://java.sun.com/jsf/composite/xpert/components"
```

Para exibir a auditoria de um determinado objeto:

```
<xc:audit for="#{personMB.person}"/>
```

Para exibir as exclusões de uma determinada classe de uma entidade:

```
<xc:auditDelete for="#{personMB.personClass}"/>
```

O objeto personClass acima deve retornar um java.lang.Class.

16.7. Ignorar auditoria de uma classe ou de um atributo

É possível ignorar auditoria para uma determinada classe ou atributo usando a anotação `@NotAudited` (com.xpert.audit), por padrão campos anotados com `@Transient` (javax.persistence) também não são auditados.

Exemplo de como ignorar auditoria de uma classe:

```
import com.xpert.audit.NotAudited;

@NotAudited
public class City {

}
```

Exemplo de como ignorar um atributo específico (nesse caso o atributo *code* será ignorado na auditoria):

```
import com.xpert.audit.NotAudited;

public class City {
    private Long id;
    private String name;
    @NotAudited
    private String code;
}
```

17. Controle de Acesso através do xpert-security

17.1. Introdução

Através desse módulo pode-se fazer o controle de acesso da aplicação. Este controle de acesso deve ser feito tanto a nível de urls, caso um usuário acesse uma url que não possua acesso ele deve ser redirecionado quanto a nível de componentes na tela, ou seja, uma dada funcionalidade só pode ser acessada para quem possuir sua permissão.

Dentro da suíte de componentes JSF também são disponibilizados componentes que fazem o controle de acesso a nível de componente (renderizar ou não um trecho da página).

Os artefatos que estão presentes no xpert-security estão listados abaixo:

AbstractUserSession

Guarda os dados da sessão do usuário. Pode ser um @ManagedBean de Sessão

AbstractSecurityFilter

Filtro para controle de usuário logado e de permissão por página.

SecurityLoginBean

Bean que gerencia o login e logout no sistema.

securityArea

Componente JSF para fazer controle de acesso a nível de componentes na tela.

17.2. Gerenciamento da Sessão a partir da classe SecuritySessionManager

A classe SecuritySessionManager (com.xpert.security) possui alguns métodos que ajudam no controle de sessão de usuário e permissões. Abaixo a lista dos principais métodos:

Método	Retorno	Descrição
clearRoles()	void	Limpa as permissões da sessão atual.
putRoles(roles)	void	Adiciona as permissões na sessão atual. O parâmetro de entrada é uma lista de do objeto com.xpert.security.model.Role.
hasURL(url)	boolean	Retorna true se a sessão atual possui a url passada por parâmetro.

hasRole(key)	boolean	Retorna true se a sessão atual possui alguma permissão com a chave passada por parâmetro. Várias chaves podem ser passadas separando por vírgula.
getRoles(request)	List<Role>	Retorna a lista de permissões da sessão atual.

17.3. Bean para manter usuário e permissões na sessão

Um usuário logado na aplicação e suas permissões devem ser mantidos na sessão, para isso é disponibilizado a classe `AbstractUserSession` (`com.xpert.security.session`), através dela é possível definir esse dados de sessão. Esta classe pode ser utilizada em combinação com o `SecurityFilter` (próximo tópico) para fazer o controle de acesso a nível de URL.

Segue abaixo um exemplo de uma classe filha de `AbstractUserSession`:


```
@ManagedBean
@SessionScoped
public class UserSessionMB extends AbstractUserSession implements Serializable {
```

```
@EJB
private RoleBO roleBO;
private AdminUser user;
private List<Role> roles;
```

```
@Override
public void createSession() {
    roles = roleBO.getRoles(user);
    super.createSession();
}
```

```
@Override
public void setUser(User user) {
    this.user = (AdminUser) user;
}
```

```
@Override
public AdminUser getUser() {
    return user;
}
```

```
@Override
public List getRoles() {
    return roles;
}
```

```
}
```

createSession() xpert-framework

Este método é utilizado para capturar as roles a partir de um usuário, a chamada do `super.createSession()` faz com que essas roles sejam colocadas na sessão. Este método faz uso da classe `SecuritySessionManager` para colocar na sessão as permissões.

setUser() xpert-framework

Deve ser utilizado para setar o usuário.

getUser() xpert-framework

Retorna o usuário da sessão atual

getRoles() xpert-framework

Retorna as permissões da sessão atual.

17.4. Filtro para bloquear o acesso do usuário por página

Para fazer esse controle por página o xpert-framework disponibiliza o filtro `AbstractSecurityFilter` (pacote `com.anuncios.filter`), essa classe é um facilitador para o controle de acesso a nível de páginas e de usuário logado na aplicação.

Exemplo de filtro para o controle de acesso:

```
@WebFilter(filterName = "AdminFilter", urlPatterns = {"/view/*"})
public class AdminFilter extends AbstractSecurityFilter {

    @Override
    public String getUserSessionName() {
        return "userSessionMB";
    }

    @Override
    public String getHomePage() {
        return "/index.jsf";
    }

    @Override
    public String[] getIgnoredUrls() {
        return null;
    }

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void destroy() {
    }
}
```

getUserSessionName()
xpert-framework
Nome do objeto de sessão que possui as informações do usuário logado. O objeto a ser guardado na sessão deve ser uma instância de AbstractUserSession (com.xpert.security.session).

getHomePage()
xpert-framework
Define a página inicial do sistema, caso não possua usuário logado, ou este não possua a página atual essa página será retornada.

getIgnoredUrls()
xpert-framework
Define as urls que não entrarão no controle de acesso. Este método retorna uma lista de urls onde não é feita a verificação de acesso.

Esse filtro é uma implementação de javax.servlet.Filter, então configurações de onde deve agir o filtro são dadas através de sua especificação.

17.5. Bean para realizar login/logout do usuário na aplicação

A classe SecurityLoginBean (com.xpert.security.bean) faz a abstração da lógica de login e logout do usuário. Ela disponibiliza métodos onde se define as configurações do login.

Exemplo de uma classe filha de SecurityLoginBean:

```
@ManagedBean
public class LoginMB extends SecurityLoginBean {
```

```
    @EJB
    private DAO dao;
    @ManagedProperty(value = "#{userSessionMB}")
    private UserSessionMB userSessionMB;
```

```
    @Override
    public String getRedirectPageWhenSucess() {
        return "/view/home.jsf";
    }
```

```
    @Override
    public String getRedirectPageWhenLogout() {
        return "/index.jsf";
    }
```

```
    @Override
    public String getUserNotFoundMessage() {
        return "User not found";
    }
```

```
    @Override
    public String getInactiveUserMessage() {
        return "User inactive";
    }
```

```
    @Override
    public EntityManager getEntityManager() {
        return dao.getEntityManager();
    }
```

```
    @Override
    public boolean isLoginUpperCase() {
        return true;
    }
```

```
    @Override
    public boolean isValidWhenNoRolesFound() {
        return true;
    }
```

```
    @Override
    public void onSuccess(User user) {
    }
```

```
    @Override
    public Class getUserClass() {
        return User.class;
    }
```

```
    @Override
    public UserSessionMB getUserSession() {
        return getUserSessionMB();
    }
```

```
    public UserSessionMB getUserSessionMB() {
        return userSessionMB;
    }
```

```
    public void setSessaoUsuarioMB(UserSessionMB userSessionMB) {
        this.userSessionMB = userSessionMB;
    }
}
```

getRedirectPageWhenSucess() xpert-framework

Define a página a ser retornada quando o login for efetuado com sucesso.

getRedirectPageWhenLogout() xpert-framework

Define a página a ser retornada quando realizado o logout.

getUserNotFoundMessage() xpert-framework

Define a mensagem a ser exibida quando o usuário não for encontrado.

getInactiveUserMessage() xpert-framework

Define a mensagem a ser exibida quando o usuário não estiver ativo.

getEntityManager() xpert-framework

Define o entity manager a ser utilizado na consulta do usuário ao se logar.

isLoginUpperCase() xpert-framework

Define se o login do sistema é uppercase, para que na verificação a String seja convertida.

isValidWhenNoRolesFound() xpert-framework

Define se deve evitado o login de um usuário que não foi encontrado permissões.

onSucess() xpert-framework

Método chamado quando o login é efetuado com sucesso, pode ser alguma lógica

Apesar de longa essa classe mostrada como exemplo exibe tudo que pode ser configurável no login do usuário, como por exemplo definir se o login deve ser UpperCase, ou ainda se deve ser validado quando não se encontra permissões para esse usuário.

17.6. Login utilizando SecurityLoginBean

Considerando a classe LoginMB (mostrada) acima a página para fazer esse login no sistema pode ser feito da seguinte maneira:

```
<h:form>
  <p:messages />
  <h:panelGrid columns="2">

    <h:outputLabel for="user" value="User:" />
    <p:inputText value="#{loginMB.userLogin}" required="true" requiredMessage="User is required"/>

    <h:outputLabel for="password" value="Password:" />
    <p:password feedback="false" value="#{loginMB.userPassword}"
      required="true" maxLength="20" requiredMessage="Password is required"/>

  </h:panelGrid>

  <p:commandButton process="@form" update="@form" value="Login" action="#{loginMB.login}" />
</h:form>
```

Esse exemplo mostrado utiliza componentes primefaces, mas eles não são obrigatórios para isso.

17.7. SecurityArea para verificação de acesso a nível de componente

Este componente está disponível através do namespace `xmlns:x="http://xpert.com/faces"` e recebe como parâmetro a chave de acesso (múltiplas chaves podem ser passadas separadas por vírgula) daquele trecho de código.

O exemplo abaixo mostra um `commandButton` que só deve ser exibido para quem possua o acesso "person.delete":

```
<x:securityArea rolesAllowed="person.delete">
  <p:commandButton process="@form" update="@form" action="#{personMB.delete}" />
</x:securityArea>
```

Abaixo o mesmo exemplo passando-se mais de uma chave:

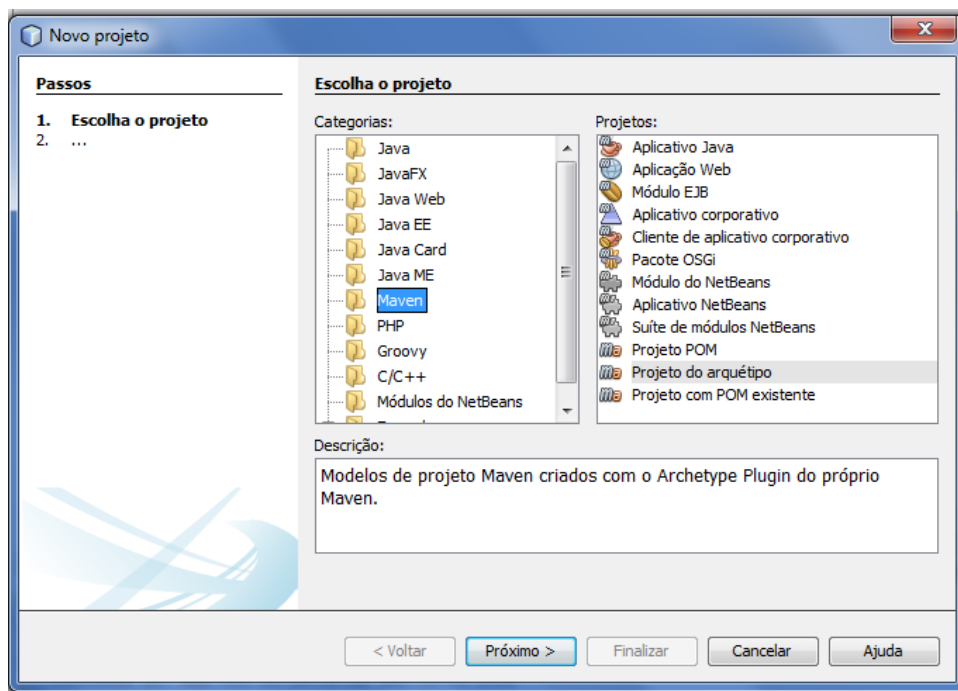
```
<x:securityArea rolesAllowed="person.delete,person.create">
  <p:commandButton process="@form" update="@form" action="#{personMB.delete}" />
</x:securityArea>
```

18. Arquétipo maven EAR do xpert-framework

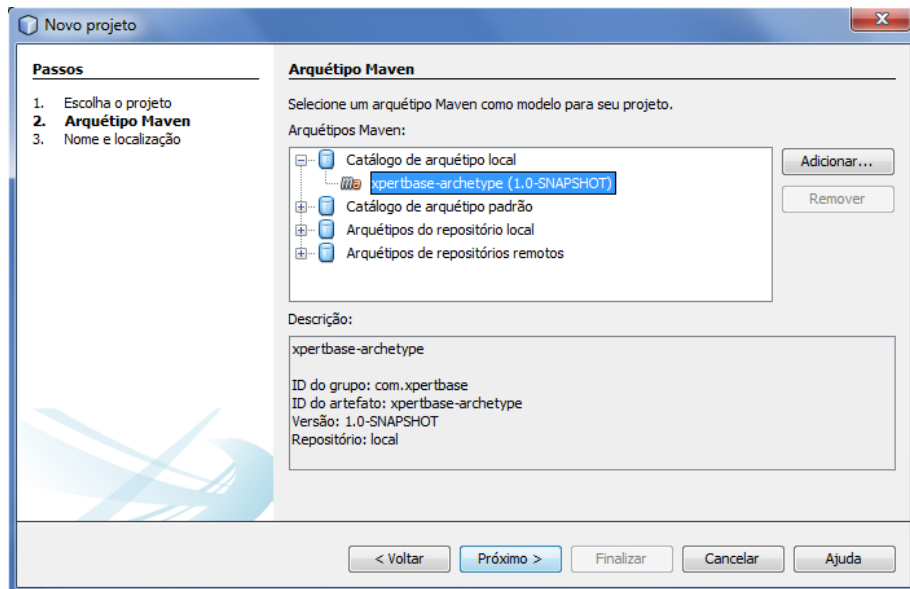
18.1. Criando um projeto a partir de um arquétipo maven com Netbeans

Para a criação de um projeto, o xpert-framework disponibiliza um arquétipo do maven para facilitar o passos iniciais do projeto.

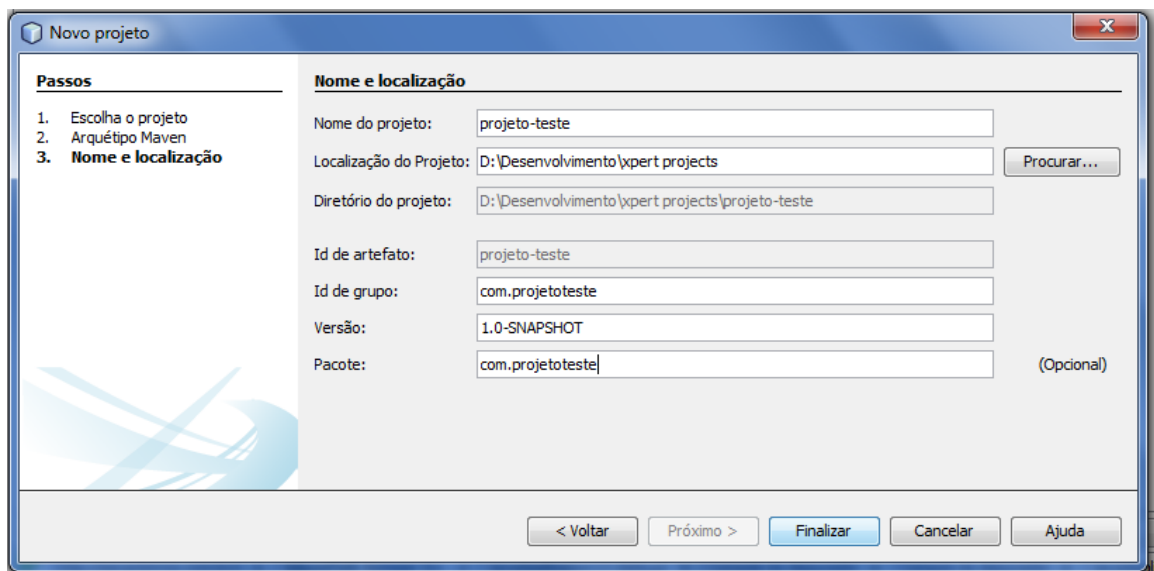
1. Caso já possua o archetype instalado na sua máquina pule para o **passo 4**, senão, baixe o projeto **xpertbase-archetype**.
2. Abra a projeto com o Netbeans: **Arquivo -> Abrir Projeto -> xpertbase-archetype**.
3. Clique com direito e selecione “Limpar e Construir”, isso faz com que o Netbeans instale o arquétipo na sua máquina.
4. Para criar o projeto vá em **Arquivo -> Novo Projeto**. Ao abrir o tipo de projeto selecione **Maven -> Projeto do Arquétipo** clique em “**Próximo**”



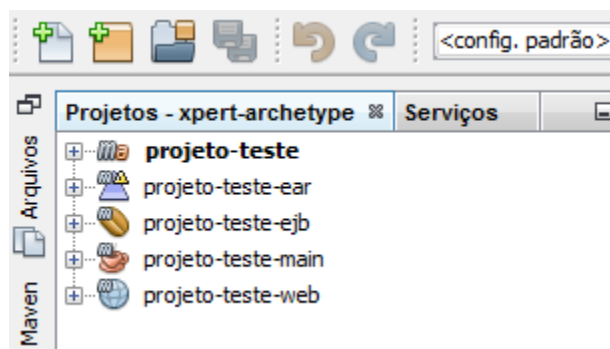
5. Expanda **Catálogo de arquétipo local** e selecione **xpertbase-archetype**, clique em “**Próximo**”:



6. Configure o projeto inserindo as informações como mostrado abaixo, e logo em seguida clique em **“Finalizar”**.



7. O projeto gerado possui a seguinte estrutura:



Isso conclui a criação de um novo EAR a partir do arquétipo maven.

Observações importantes:

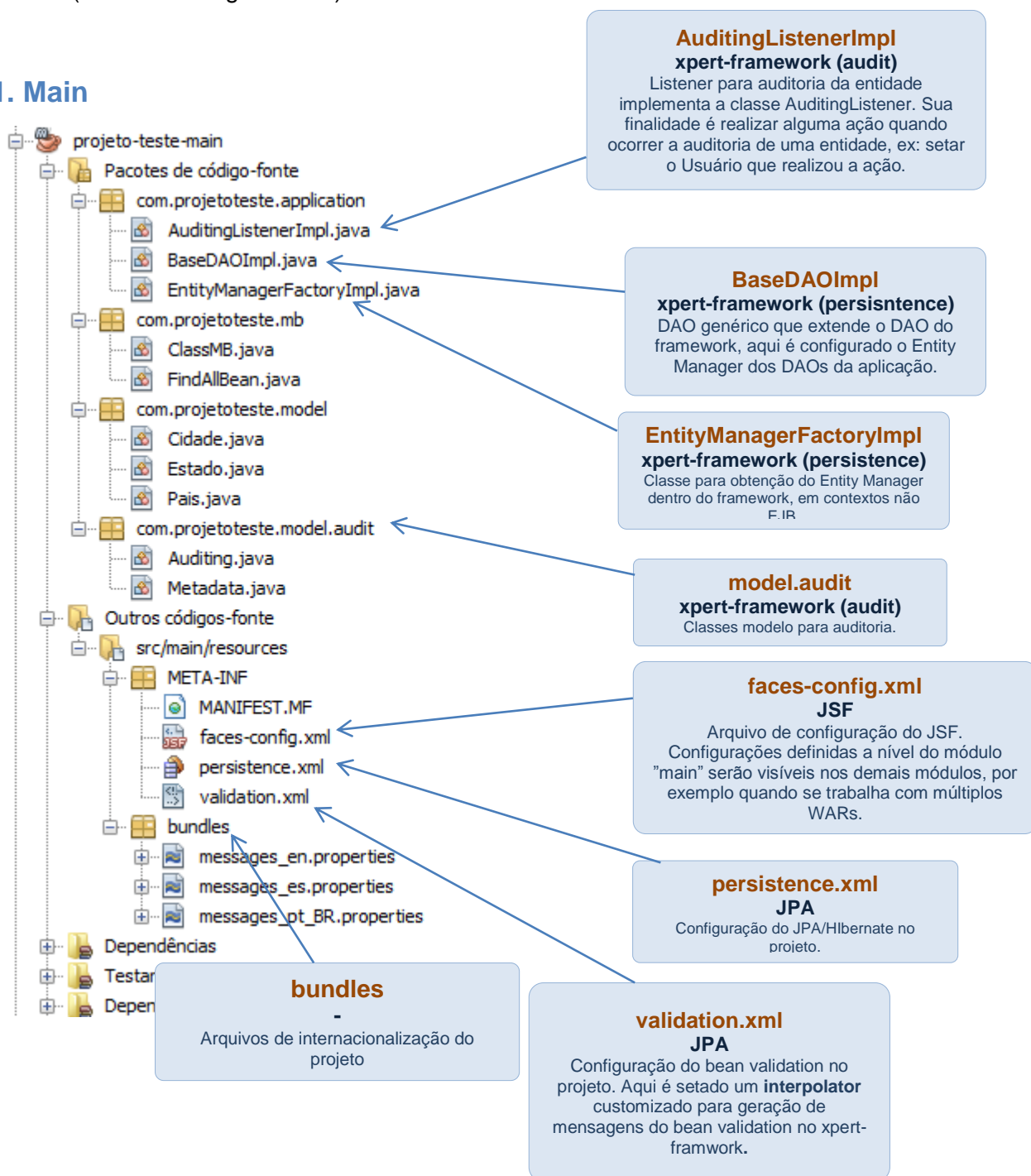
- O arquétipo pode possuir referência a uma versão antiga de alguma lib, como por exemplo, do próprio xpert-framework, para resolver o problema abra o arquivo pom.xml do projeto **main** e muda para a versão desejada.

18.2. Estrutura de um projeto gerado a partir do arquétipo maven

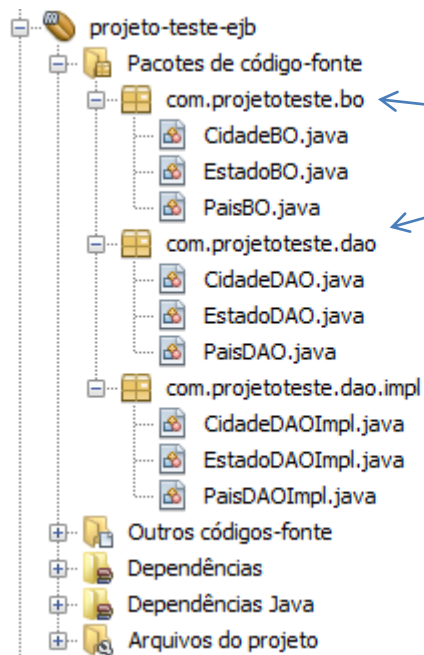
O projeto gerado se divide em 4 módulos:

- **EAR**
- **main** (este projeto possui configurações comuns a todos os módulos, como o **persistence.xml**, **validation.xml**, arquivos de **i18n** e as **classes modelo**)
- **EJB** (BOs e DAOs)
- **WAR** (xhtml e ManagedBeans)

18.2.1. Main



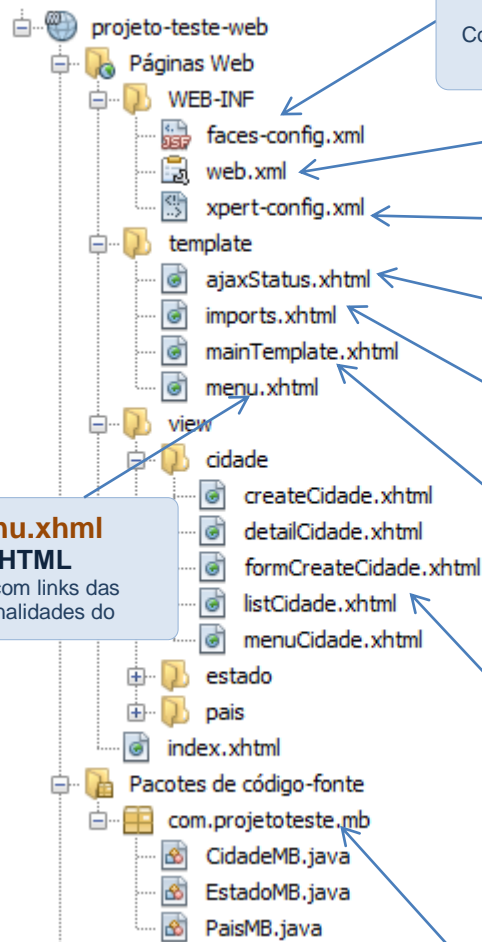
18.2.2. EJB



BO, DAO, DAOImpl

Essas classes são Session Beans (@Stateless) por isso devem ficar nesse módulo. O DAO é apenas a interface (anotado com @Local).

18.2.3. WAR



faces-config.xml JSF

Arquivo de configuração do JSF. Configurações definidas aqui serão visíveis apenas nesse WAR.

web.xml JEE

Arquivo descritor padrão de uma aplicação web.

xpert-config.xml xpert-framework

Arquivo de configuração do xpert-framework.

ajaxStatus.xhtml XHTML

Arquivo para ser usado na forma de **include** para definir algum tipo de bloqueio de tela para requisições ajax. Possui um **p:ajaxStatus** (primefaces).

imports.xhtml XHTML

Imports de CSS e javascripts comuns a toda aplicação.

mainTemplate.xhtml XHTML

Template genérico da aplicação. Aqui é feito include do **imports.xhtml**, **menu.xhtml** e do **ajaxStatus.xhtml**

menu.xhtml XHTML

Menu com links das funcionalidades do

mb

@ManagedBean - JSF

Pacote que possui os Managed Beans do Projeto.

Exemplo de CRUD

Um CRUD gerado pelo maker possui essa estrutura.