**lindbergframework – Módulo Validation**

**Conteúdo**

[1. Introdução](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#intro)

[2. Iniciando com as validações](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#inic)

[3. Modos de execução de validações](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#modod)

[4. Dependências](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#dep)

[5. Validações na prática](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#valid)

[5.1. Exemplo 1 – Validações usando annotations](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#ex1)

[5.2. Exemplo 2 – Validações usando ações via annotations](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#ex2)

[5.3. Exemplo 3 – Validações usando Item´s](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#ex3)

[5.4. Exemplo 4 – Validações usando Item´s](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#ex4)

[5.5. Execução de validações definindo o modo de processamento](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#exec)

[5.6. Adicionar validações validando](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#addval)

[6. Como estender o contexto de validações do lindbergframework](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#comoest)

[7. Como usar validações externas (criadas por você) para definir validações via annotations do lindbergframework](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#comousar)

[8. Projeto Exemplo de uso do lindbergframework](file:///C:\Users\Lindberg\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.IE5\E3E5VDUK\framework%5b1%5d.htm#projexe)

**1. Introdução**

O validation do lindbergframework provê uma forma elegante e eficiente de tratar validações. A idéia é escrever as cada validação apenas uma vez isolando-a de modo a reutilizar essas validações em todos os módulos do projeto e em outros projetos quando necessário.

O framework é extensível, possibilitando uma equipe que faz uso dele a criação de suas próprias validações, a adição de novas funcionalidades, levando os projetos que o utilizam a terem seu próprio repositório de validações para uso em todos os projetos. Isso tudo contando com o suporte do lindbergframework para o processamento dessas validações tanto via código direto quanto via annotation.

O cerne principal do módulo validation do framework é fornecer uma forma criativa, prática, elegante e eficitente validações como CPF, campo requerido, datas, etc... As validações e seus tipos são ilimitadas indo desde validações menores como citadas anteriormente a validações mais específicas de regras de negócio como por exemplo a possibilidade de uma confirmação de um emprestimo financerio.

É uma constante no processo de desenvolvimento de aplicações corporativas a implementação de regras de negócio e muitas vezes precisamos verificar diversas validações em uma ação como por exemplo “confirmar”. A forma como devem ser feitas essas validações bem como o padrão das mensagens tem uma gama de possibilidades de ação para ação, de regra para regra. Muitas vezes, deve-se aninhar as validações, em outros momentos lança-las imediatamente, em outras vezes talvez tenhamos que concatenar a mensagem de exceção ou validação com uma mensagem customizada, entre outras. Isso tudo faz com que em muitas vezes o código perca a organização, a legibilidade, fuja de um padrão definido ou mesmo nem tenha um sendo comum ver uma legião de “if's” no código para efetuas as mais variadas verificações. Essa série de fatores faz com que a manutenabilidade e a evolução do código se tornem difíceis.

O lindbergframework é uma solução que padroniza e provê diversos recursos para resolver esses problemas e de uma forma simples que você vai dizer “como é que eu nunca pensei nisso antes?”.

**2. Iniciando com as validações**

As validações dentro do framework seguem as seguintes configurações não sendo nenhuma requerida:

**a.** Mensagem customizada para validação;

**b.** Tipo de uso da mensagem de validação.

Esse tipo é baseado no enum MsgType que define os tipos possíveis

MsgType:

\* *CUSTOM\_ONLY* *: Usa* *somente* *a* *mensagem* *personalizada*

\* *USING\_CUSTOM\_SUFFIX* *: Usa* *a* *mensagem* *personalizada* *como* *sufixo* *na* *mensagem* *de* *validação*

\* *USING\_CUSTOM\_PREFIX : Usa* *a* *mensagem* *personalizada* *como* *prefixo* *na* *mensagem* *de* *validação*

\* *NO\_USING\_CUSTOM : Usa* *apenas* *a* *mensagem* *de* *validação* *e* *não* *considera* *a* *personalizada*

**c.** Separator (SEPARADOR) das mensagens provenientes da validação (ValidationException) e a mensagem customizada. O separator é uma String que comporá a mensagem final da validação. Ele é usado quando a mensagem da exception e a mensagem customizada são usadas em conjunto de modo que essa String;

**d.** Actions: ações que estão associadas a validação.

Um objeto *Item*corresponde ao objeto que queremos validar. Este objeto possui as configurações citadas acima que são definidas pelo desenvolvedor e juntas compõem a regra. O mecanismo de validação do framework funciona da seguinte forma: sempre um item de validação é criado para cada propriedade ou objeto a ser validado e este item está sempre associado a uma ou mais validações. Uma validação dentro do framework corresponde a uma classe que implemente a interface Ivalidation. Uma validação deve implementar o método validate que é definido na interface IValidation e neste método deve estar a regra ao qual a validação se propõe a validar. Sempre funciona dessa forma: temos um item que corresponde a um objeto a ser validado, que contem também as definições das customizações que devem ser aplicadas as validações e associamos validações a este item. Este processo se repete o número de vezes que for necessário em um mesmo método.

O processo de validação é gerenciado e executado por um objeto que implemente a interface IExecutorValidation. Um executor que implemente esta interface fornece os métodos e recursos necessários para a composição de toda a regra de validação de uma ou mais entidades, em um ou mais eventos.

O lindbergframework fornece 4 implementações que estão no pacote *org.lindbergframework.validation.executors*, descritas abaixo, de executors que devem ser usados de acordo com a necessidade:

**ExecutorValidationItemsImpl** : Executor padrão para casos que não utilizem annotations

**ExecutorValidationItemMultThread**: Executor multthread para casos que não utilizem annotations e que são validações que exigem um volume maior de processamento. Neste executor cada validação é processada separadamente em uma thread.

**ExecutorAnnotationEngineImpl**: Executor padrão para casos que utilizam annotations

**ExecutorAnnotationEngineMultThread**: Executor multhread para casos que utilizam annotations e que são validações que exigem um volume maior de processamento. Neste executor cada validação é processada separadamente em uma thread.

O framework fornece factories para a obtenção de instancias das interfaces definidas. As implementações das validações e dos executors não devem ser acessadas diretamente. Ao invés disso, use as interfaces e as factories para a obtenção das implementações:

**ExecutorFactory**: Fábrica de Executors;

**ValidationFactory**: Fábrica de Validações.

**3. Modos de execução de validações**

O processo de execução das validações pode ser realizado de 2 modos principais.  O modo padrão como é feita a validação é o de lançar as exceções, caso haja alguma, apenas ao final da execução. O modo pode ser configurado no momento da chamada do método execute() de um executor que suporte o parâmetro mode. Caso o método execute chamado não seja o que possui o parâmetro mode então o modo de execução assumido é o padrão. O modos estão definidos pelo enum ValidationMode. Abaixo estão descritos os dois modos de execução:

**THROW\_FINAL**: Lança exceções no final aninhando as exceções. Este modo indica que um executor validation vai aninhar as mensagens as exceções que ocorrerem na validação e lançar apenas no final da validação caso tenha havido alguma falha de alguma validação;

**THROW\_IMMEDIATELY**: Lança exceções imediatamente não aninhando as exceções. Este modo indica que um executor validation não vai aninhar as exceções ou seja que assim que alguma validação falhar será lançada uma exceção.

**4.****Dependências**

O lindbergframework tem algumas dependências que devem ser adicionadas junto com a lib do framework ao projeto que use o framework. Cada dependência é usada pelo lindbergframework e não obriga ou condiciona um projeto que use o framework a usar qualquer das bibliotecas das quais o lindbergframework depende. Abaixo está a lista de dependências do framework. Para baixar as dependências do framework acesse o item de menu *downloads.*

\* spring-2.5.5.jar

\* commons-lang-2.4.jar

\* commons-beanutils-1.7.0.jar

**\***commons-logging-1.1.1.jar

**5. Validações na prática**

Vamos considerar o bean abaixo para iniciar as validações.

/\*\*

 \*

 \* **@author** Victor Lindberg

 \*

 \*/

**public** **class** Pessoa {

      //requerido

**private** String nome;

**private** String cpf;

        //não é requerido mas se for informado não permite informação vazia

**private** String endereco;

        //requerido e a data não pode ser futura

**private** Date dtNascimento;

        //não pode ser menor do que a data atual

**private** Date dtValidadeCadastro;

       //deve conter pelo menos 1 telefone

**private** List<String> telefones;

**public** Pessoa(){

                //

        }

        //setters e getters

}

Seguindo as validações definidas nos comentários serão demostrados vários exemplos usando este bean tanto com annotations quanto diretamente via objetos *Item*.

**5.1. Exemplo 1 – Validações usando annotations**

O bean acima ainda só tem as validações via comentário que devem ser realizadas.  O lindbergframework define 2 annotations para a definição de validações: @Valid e @Validations. A primeria define uma annotation em si. A segunda define um conjunto de annotations. A annotation @Valid é usada para definir uma validação e configuração específicas para um campo. Devido a impossibilidade de repetição da mesma annotation para um campo a annotation @Validations é fornecida para a definição de várias @Valid para um mesmo campo.

Vamos então anotar o bean, abaixo, usando as annotations do lindbegframework para configurar as validações descritas nos comentários do bean.

**import** java.util.Date;

**import** java.util.List;

**import** java.util.Vector;

**import** org.lindbergframework.validation.Types;

**import** org.lindbergframework.validation.annotation.Valid;

**import** org.lindbergframework.validation.annotation.Validations;

**import** org.lindbergframework.validation.settings.MsgType;

/\*\*

 \*

 \* **@author** Victor Lindberg

 \*

 \*/

**public** **class** Pessoa {

        @Valid(value = Types.*REQUIRED\_FIELD*, msg = "nome é requerido")

**private** String nome;

        @Valid(Types.*CPF*)

**private** String cpf;

        //não é requerido mas caso seja informado não permite informação vazia (espaços em branco).

        //A mensagem dessa validação é concatenada com a mensagem da validação senda a mensagem

     //informada aqui sufixo da mensagem da validação

        @Valid(value = Types.*NO\_INFORMATION\_EMPTY*, msg = "endereço", msgType = MsgType.*USING\_CUSTOM\_SUFFIX*)

**private** String endereco;

        @Validations({@Valid(value = Types.*REQUIRED\_FIELD*,msg = "é necessário informar data de nascimento",

                                msgType = MsgType.*USING\_CUSTOM\_SUFFIX*),

                          @Valid(value = Types.*DATE\_CAN\_NOT\_BE\_FUTURE*, msg = "data de nascimento",

                                msgType = MsgType.*USING\_CUSTOM\_SUFFIX*)})

**private** Date dtNascimento;

        //nesta validação a mensagem é concatenada com da validação sendo a definida aqui

      //prefixo da mensagem da validação

        @Valid(value = Types.*DATE\_CAN\_NOT\_BE\_PAST*, msg = "data validade cadastro",

                        msgType = MsgType.*USING\_CUSTOM\_PREFIX*,separator = "-->")

**private** Date dtValidadeCadastro;

        @Valid(value = Types.*NOT\_EMPTY\_LIST*,msg = "nenhum telefone informado")

**private** List<String> telefones;

**public** Pessoa(){

                //

        }

**public** **static** Pessoa getPessoaComCamposInvalidosParaTeste(){

                Pessoa pessoa = **new** Pessoa();

                //o nome é requerido então validação de "campo requerido" vai falhar

                pessoa.setNome(**null**);

                //cpf inválido então validação de cpf vai falhar

                pessoa.setCpf("12345678900");

                //endereço não é requerido mas se for informado não aceita informação

             //vazia (espaços em branco)

                pessoa.setEndereco("   ");

                //data de nascimento maior do que a atual então validação de "data não

              //pode ser futuro" vai falhar

                pessoa.setDtNascimento(**new** Date(3000,1,1));

                //data de validade de cadastro não pode ser menor do que a atual então validação

             //de "data não pode ser passado" vai falhar

                pessoa.setDtValidadeCadastro(**new** Date(1,1,1));

                //lista de telefones deve conter pelo menos um número então validação

             //"lista não pode ser vazia" vai falhar

                pessoa.setTelefones(**new** Vector<String>());

**return** pessoa;

        }

        //setters e getters

}

Para evitar o grande volume do texto o bean está todo comentado esclarecendo cada trecho de código.

O exemplo abaixo executa uma validação de um objeto Pessoa. As regras de validação para o objeto Pessoa foram anotadas no bean usando as annotations **@Valid** e **@Validations**.

///////Popula o bean

Pessoa pessoa = Pessoa.getPessoaComCamposInvalidosParaTeste();

////// Executa a validação

//observe que a interface que define os engines de processamento de annotations é IExecutorAnnotationEngine

//cria executor de beans anotados

IExecutorAnnotationEngine executor = ExecutorFactory.newExecutorAnnotationEngine();

executor.execute(pessoa);

No caso acima é utilizado um executor padrão para efetuar a validação. Observe que a interface usada é a IexecutorAnnotationEngine e não IExecutorValidation.

Isso ocorre pelo fato de IexecutorValidation definir um tipo geral de executor. Neste caso precisamos de executor específico para annotations então usamos IexecutorAnnotationEngine que é uma interface que extende da interface IExecutorValidation.

O método execute processará as regras definidas via annotation no bean pessoa e como não foi especificado nenhum modo de validação específico (ValidationMode) então o padrão é usado: que é o aninhamento de exceções ou seja efetua todas as validações e lança todas as que falharam apenas no final do processamento de toda a validação. O lançamento das exceções, caso haja alguma, é feito lançando-se uma ValidationException contendo todas as mensagens devidamente configuradas das exceções que falharam.

**5.2. Exemplo 2 – Validações usando ações via annotations**

O lindbergframework fornece um mecanismo que possibilita definir as validações dos beans e especificar em quais ações cada validação será processada.

Por exemplo podemos ter 3 campos em um bean onde 2 são validados quando a ação for X e 1 quando a ação for Y. Outro exemplo seria um bean com 3 campos onde 2 são validados de forma diferente de acordo com a ação. Quando a ação for X valida de um jeito mas quando a ação for Y valida de outro jeito.

Este bean esta anotado definindo ações para cada validação. Na hora de executar as validações usando uma implementação de IExecutorAnnotationEngine fazendo a execução através do método executeInActions só serão processadas as validações que estejam relacionadas a no mínimo uma das ações passadas como parametro para o método executeInActions. Abaixo é mostrado um bean para exemplificar o uso das ações no processo de validação.

**import** org.lindbergframework.validation.Types;

**import** org.lindbergframework.validation.annotation.CrudActions;

**import** org.lindbergframework.validation.annotation.Valid;

/\*\*

 \*

 \* **@author** Victor Lindberg

 \*

 \*/

**public** **class** BeanTesteActions {

        //Bean para demostração de definição de validações de acordo com ações.

        //essa validação só será usada quando a ação for "cadastrar" ou quando a execução

      //não for feita pelo método executeInActions de IExecutorAnnotationEngine

      //ou nenhuma ação for definida no momento da execução

        @Valid(value = Types.*NOT\_NULL*,msg = "falha campo1", actions = "cadastrar")

**public** String campo1;

        //essa validação só será usada quando a ação for "cadastrar" ou "qualquerAcao" ou

      //quando a execução não for feita pelo método executeInActions de IExecutorAnnotationEngine

      //ou nenhuma ação for definida no momento da execução

        @Valid(value = Types.*NOT\_NULL*, msg = "falha campo2",actions = {"consultar", "qualquerAcao"})

**public** String campo2;

        //essa validação só será usada quando a ação for "cadastrar", "excluir" ou "qualquerAcao"

      //ou quando a execução não for feita pelo método executeInActions de IExecutorAnnotationEngine

      //ou nenhuma ação for definida no momento da execução

        @Valid(value = Types.*NOT\_NULL*, msg = "falha campo3",

                actions = {"cadastrar", "excluir", "qualquerAcao"})

**public** String campo3;

        //essa validação usa uma interface de constantes fornecida pelo framework para padronizar e

        //facilitar a referencia a ações do tipo CRUD

        @Valid(value = Types.*NOT\_NULL*, msg = "falha campo4",actions = CrudActions.*UPDATE*)

**public** String campo4;

        //essa validação não esta relacionada a nenhuma ação então só será executada

       //quando a execução não realizada referenciando nenhuma ação específica

        @Valid(value = Types.*NOT\_NULL*, msg = "falha campo5")

**public** String campo5;

**public** BeanTesteActions(){

                //

        }

}

Observe que cada propriedade do bean está comentada de forma a esclarecer a configuração definida para a validação da propriedade.

OBSERVAÇÃO: Veja que todos os campo são public, isto foi feito de propósito para demostrar que o framework verifica se o campo está acessível, se estiver o acessa diretamente caso não esteja tentar invocar um método get do campo seguindo a especificação JAVABEAN. Por exemplo um campo "private nome" o framework procura getNome. Abaixo é exibido um exemplo simples de validação usando ações.

BeanTesteActions beanTesteActions = **new** BeanTesteActions();

////// Executa a validação

//observe que a interface que define os engines de

//processamento de annotations é IExecutorAnnotationEngine

ExecutorAnnotationEngine engine = ExecutorFactory.newExecutorAnnotationEngine();

//aqui foi usado o método addBeans para adicionar beans anotados quando quiser antes de chamar execute()

engine.addBeans(beanTesteActions);

//executa as validações associadas a ação de "cadastrar".

//No caso do bean em questão (BeanTesteActions) apenas a validação do campo 1 e 3

//veja as anotações do bean BeanTesteActions para verificar que apenas os campos 1 e 3 tem

//validações associadas a action "cadastrar"

engine.executeInActions("cadastrar");

Observe que o método executeInActions foi utilizado para processar as regras de validações. Esse método especifica que deve ser levado em consideração as ações e define quais ações.

Neste caso foi especificada apenas a ação de “cadastrar” definindo que apenas as validações associadas a ação de “cadastrar” serão processadas. O parâmetro deste método é um vargars ou seja String... então pode ser passada uma quantidade variável de ações para na mesma chamada deste método fazendo com que sua assinatura seja flexível. Abaixo é mostrado um exemplo da definição de várias ações para o mesmo processo de validação.

BeanTesteActions beanTesteActions = **new** BeanTesteActions();

////// Executa a validação

//observe que a interface que define os engines de processamento de annotations é IExecutorAnnotationEngine

IExecutorAnnotationEngine engine = ExecutorFactory.newExecutorAnnotationEngine();

//aqui foi usado o método addBeans para adicionar beans anotados quando quiser antes de chamar execute()

engine.addBeans(beanTesteActions);

//Neste exemplo executa as validações associadas a ação de "cadastrar", "qualquerAcao"

//e definida na constante Types.UPDATE da interface CrudActions.

//No caso do bean em questão (BeanTesteActions) apenas a validação do campo 1,2,3 e 4

engine.executeInActions("cadastrar","qualquerAcao",CrudActions.UPDATE);

No exemplo acima apenas as validações que estão associadas as ações “cadastrar”, “qualquerAcao” e a constante UPDATE  da interface CrudActions são processadas.

A interface CrudActions é definida de modo a forncer constantes pré-definidas para as ações CRUD. Usando essa interface a definição de ações crud nas validações se torna padronizada, centralizada nesta interface e evita a inserção manual de String eliminando os erros decorrentes de digitação.

Observe que no exemplo acima o bean a ser validado foi definido através do método addBeans. Esse método recebe também um varargs de object (Object...) de modo que é possível adiciona qualquer quantidade de beans na mesma chamada de método como exemplificado abaixo:

BeanTesteActions beanTesteActions = **new** BeanTesteActions();

Pessoa pessoa = Pessoa.getPessoaComCamposInvalidosParaTeste();

////// Executa a validação

//observe que a interface que define os engines de processamento de annotations é IExecutorAnnotationEngine

IExecutorAnnotationEngine engine = ExecutorFactory.newExecutorAnnotationEngine();

//adiciona 2 beans anotados para validação

engine.addBeans(beanTesteActions, pessoa);

engine.execute("\n");

No exemplo acima observe que o método execute usado foi o com 1 parametro String. Esse método executa a validação definindo o separadaor de mensagens, ou seja a String que vai separar uma mensagem de validação da outra, como sendo um '\n' para pular linha. Com esse separador cada mensagem de validação ficará em uma linha no console. OBSEVAÇÃO: A String do separador pode ser qualquer uma. Caso um separador não seja definido o separador padrão, LindbergException.DEFAULT\_SEPARATOR\_MESSAGES ,é usado. O separador estará presente quando o método getMessageTreated(), toString() ou getMessage() da ValidationException originada do processamento das validações, casoalguma falhe, seja chamado.

**5.3. Exemplo 3 – Validações usando Item´s**

Uma outra forma de definir validações é a criação de regras de validação que ao invés de usar annotations usa Objetos *Item*. A classe*Item* suporta as configurações, tópico 2 deste documento, necessárias para a definição das validações. Um objeto item encapsula um Objeto a ser validado junto com as configurações customizadas para a validação do objeto. Abaixo é mostrado um exemplo da mesma validação da classe pessoa que foi feita via annotations só que sendo feita diretamente fazendo uso de *Items*.

 Pessoa pessoa = Pessoa.getPessoaComCamposInvalidosParaTeste();

//Aqui não estamos utilizando validações via annotations, estamos usando

//via items diretamente então precisamos de uma implementação da interface IExecutorValidationItems

IExecutorValidationItems executor = ExecutorFactory.newExecutorValidationItems();

//definindo as validações diretamente

//validação de nome requerido

executor.addValidationsForItem(**new** Item(pessoa.getNome(),"nome é requerido"),

                ValidationFactory.newRequiredField());

//validação de cpf

executor.addValidationsForItem(**new** Item(pessoa.getCpf()),

                ValidationFactory.newCpf());

//validação de endereço se informado não pode ser vazio

executor.addValidationsForItem(**new** Item(pessoa.getEndereco(),MsgType.USING\_CUSTOM\_SUFFIX,"endereço"),

                ValidationFactory.newNoInformationEmpty());

//validação de "data de nascimento requerida" e de "data não pode ser futura"

//pois uma pessoa não pode ter data de nascimento maior do que a atual

executor.addValidationsForItem(**new** Item(pessoa.getDtNascimento(),MsgType.USING\_CUSTOM\_SUFFIX,

                "é necessário informar data de nascimento","data de nascimento"),

                ValidationFactory.newRequiredField(),

                ValidationFactory.newDateCanNotBeFuture());

//validação de data de cadastro

//Essa validação define um separador "-->" que será a String entre a mensagem

//da exception e a mensagem definida aqui.

//A concatenação da mensagem da exception e da mensagem definida aqui se

//dará através do tipo MsgType.USING\_CUSTOM\_PREFIX

//que usa a mensagem definida aqui como prefixo da mensagem da exception no seguinte formato:

//mensagem definida aqui --> mensagem da exception

executor.addValidationsForItem(**new** Item("-->",pessoa.getDtValidadeCadastro(),

                                        MsgType.USING\_CUSTOM\_PREFIX,"data validade cadastro"),

                               ValidationFactory.newDateCanNotBePast());

//validação de nro de telefones que deve ser fornecido ao menos um

//OBS: Aqui não valida se a lista é nuul ou não apenas se está vazia

executor.addValidationsForItem(**new** Item(pessoa.getTelefones(),"nenhum telefone informado"),

                ValidationFactory.newNotEmptyList());

executor.execute();

Cada trecho de código do exemplo acima que adiciona os Items está comentado e descrevendo o que foi feito. Observe que a interface que foi utilizada foi a IExecutorValidationItems que é a interface que define os Executors que processam validações definidas usando Objetos*Item* diretamente.

A validação da data de cadastro exemplifica a definição de separadores e da concatenação da mensagem de validação com a customizada de acordo com o Enum MsgType. No caso dessa validação é definido um separador ‘-->’ e o uso da mensagem customizada como sendo PREFIXO (MsgType.USING\_CUSTOM\_PREFIX) da mensagem da validação. Isso configura a mensagem resultante de falha da validação no seguinte padrão ‘mensagem customizada --> mensagem da exception’.

Uma definição importante é a da validação da data de nascimento que usa um mesmo método *addValidationsForItem* do executor para efetuar a associação de 2 validações para o Item que encapsula o objeto data de nascimento. Quando não for definida nenhuma mensagem o MsgType NO\_USING\_CUSTOM é usado configurando o uso apenas da mensagem da ValidationException. Da mesma forma quando o separador não é informado o separador padrão é usado, Item.DEFAULT\_SEPARATOR.

**5.4. Exemplo 4 – Validações usando Item´s**

Existem várias formas e variações possíveis para se montar as validações. Para tal é bom escolher qual método se adéqua melhor a validação que se deseja adicionar ao Executor, por isso é bom navegar pela interface IExecutorValidationItems. Abaixo é mostrado um exemplo dessa flexibilidade efetuando a mesmas validações do exemplo do tópico anterior só que de outra forma.

              Pessoa pessoa = Pessoa.*getPessoaComCamposInvalidosParaTeste*();

                //Aqui não estamos utilizando validações via annotations, estamos usando

                //via items diretamente então precisamos de uma implementação da interface IExecutorValidationItems

                IExecutorValidationItems executor = ExecutorFactory.*newExecutorValidationItems*();

                //adiciona as validações. Essa forma de adição de validações

                //associa uma série de items a uma série de validações na mesma chamada do

             //método addValidations. Como então o validador sabe quais validações usar

             //para cada item? Aqui são usados os construtores de objetos Item que recebe integer's

             //indicando os índices das validações que o item requer para ser validado levando em consideração

                //que nesta mesma chamada de método a validação de índice 0 é a primeira

             //(neste caso ValidationFactory.newRequiredField()),

                //a de índice 1 a segunda e assim por diante até a ultima (ValidationFactory.newNotEmptyList())

             //que neste caso tem índice 5.

                executor.addValidations(ArrayUtil.*toArray*(

                        ///////////items a serem validados

                    //vai usar apenas a validação de índice 0, ValidationFactory.newRequiredField()

**new** Item(pessoa.getNome(),"nome é requerido",0),

                    //vai usar a validação de índice 1 e apenas essa

**new** Item(pessoa.getCpf(),1),

**new** Item(pessoa.getEndereco(),"endereço",MsgType.*USING\_CUSTOM\_SUFFIX*,2),

                        //o item abaixo vai usar as validações de índice 0 e 3. ESte exemplo ilustra

                    //que o limite das validações a serem associadas a cada item são as validações

                    //que participam da mesma chamada de método

**new** Item(pessoa.getDtNascimento(),MsgType.*USING\_CUSTOM\_SUFFIX*,

                                        ArrayUtil.*toArray*("é necessário informar data de nascimento",

                                       "data de nascimento"),0,3),

**new** Item("-->",pessoa.getDtValidadeCadastro(),"data validade cadastro",

                                       MsgType.*USING\_CUSTOM\_PREFIX*,4),

**new** Item(pessoa.getTelefones(),"nenhum telefone informado",5)),

                                //////////validações

                                /\*0\*/ValidationFactory.*newRequiredField*(),

                                /\*1\*/ValidationFactory.*newCpf*(),

                                /\*2\*/ValidationFactory.*newNoInformationEmpty*(),

                                /\*3\*/ValidationFactory.*newDateCanNotBeFuture*(),

                                /\*4\*/ValidationFactory.*newDateCanNotBePast*(),

                                /\*5\*/ValidationFactory.*newNotEmptyList*());

                //executa a validação só que definindo um separador diferente para melhor

                  //visualização de onde se localiza um separador na mensagem

                  executor.execute(" (Aqui é um ; ) ");

Acima é usado o método *addValidations* para adicionar as validações. Esse método funciona adicionando na mesma chamada de método vários Item de validação associando cada um as validações requeridas para a correta validação das regras. O método *addValidations*recebe 2 parametros, sendo o primeiro um array de objetos Item onde cada Item representa um Objeto a ser validado e as configurações customizadas para a validação do objeto, e o segundo parâmetro um varargs de IValidation (IValidation...) com as validações que podem ser usadas ou não por cada Item. O construtor usado para criar cada Item neste caso é o que recebe um varargs de Integer (Integer...) indicando os índices das validações que devem ser usadas para validar o item. O índice de cada validação é definido pela sua posição na chamada de método sendo no exemplo acima a *newRequiredField* a de índice 0, *newCpf* índice 1, etc...

Observe que o método *execute()* usado foi passando um separator “ (Aqui é um ; )  “ mostrando que qualquer String pode ser definida como separator.  O exemplo acima está todo comentado trecho por trecho, leia os comentários com atenção para um melhor esclarecimento. Abaixo o mesmo exemplo é mostrado sem os comentários para uma melhor visualização do exemplo.

Pessoa pessoa = Pessoa.*getPessoaComCamposInvalidosParaTeste*();

IExecutorValidationItems executor = ExecutorFactory.*newExecutorValidationItems*();

//Fizemos toda a validação do ExemploValidacaoUsandoItem1 com apenas uma chamada de método

executor.addValidations(ArrayUtil.*toArray*(

**new** Item(pessoa.getNome(),"nome é requerido",0),

**new** Item(pessoa.getCpf(),1),

**new** Item(pessoa.getEndereco(),"endereço",MsgType.*USING\_CUSTOM\_SUFFIX*,2),

**new** Item(pessoa.getDtNascimento(),MsgType.*USING\_CUSTOM\_SUFFIX*,

                                                ArrayUtil.*toArray*("é necessário informar data de nascimento",

                                                          "data de nascimento"),0,3),

**new** Item("-->",pessoa.getDtValidadeCadastro(),"data validade cadastro",

                                        MsgType.*USING\_CUSTOM\_PREFIX*,4),

**new** Item(pessoa.getTelefones(),"nenhum telefone informado",5)),

                                /\*0\*/ValidationFactory.*newRequiredField*(),

                                /\*1\*/ValidationFactory.*newCpf*(),

                                /\*2\*/ValidationFactory.*newNoInformationEmpty*(),

                                /\*3\*/ValidationFactory.*newDateCanNotBeFuture*(),

                                /\*4\*/ValidationFactory.*newDateCanNotBePast*(),

                                /\*5\*/ValidationFactory.*newNotEmptyList*());

**5.5. Execução de validações definindo o modo de processamento**

O modo de execução das validações pode ser configurado de acordo com a necessidade. O padrão é aninhar as exceções ou seja processar cada validação definida juntar tudo e lançar as que falharam no final, mas existe também a possibilidade de lançar assim que uma das validação falhar. Para tal basta configurar o modo de execução da seguinte forma:

   executor.execute(ValidationMode.THROW\_IMMEDIATELY);

**5.6. Adicionar validações validando**

É possível não usar o método *execute.* Para tal os método usados para adicionar os items de validação são todos sobrecarregados com uma opção com o sufixo Validating que entre os parâmetros normais recebe um ValidationMode para a definição do modo de execução de validações. Abaixo é mostrado um exemplo:

Os método abaixo são definidos na interface ***IExecutorValidationItems***

/\*\*

 \* Adiciona várias validações a um item

 \*/

**public** **void** addValidationsForItem(Item item,IValidation... validacoes);

/\*\*

 \* Adiciona várias validações a um item e

 \* já efetua a validação no modo {@link ValidationMode.THROW\_FINAL}

 \*/

**public** **void** addValidationsForItemValidating(Item item,IValidation... validacoes);

/\*\*

 \* Adiciona várias validações a um item e

 \* já efetua a validação no modo que é passado como argumento para o método no parametro mode

 \*/

**public** **void** addValidationsForItemValidating(ValidationMode mode, Item item,IValidation... validacoes);

**6. Como estender o contexto de validações do lindbergframework**

O lindbergframework usa o framework spring para gerenciar o contexto dos beans validations criados. Isso não obriga você a usar o spring. O contexto de validações é o local onde estão os beans validations (objetos que implementam IValidation) que o lindbergframework fornece já prontos para uso. Uma das idéias de um framework é a possibilidade de extensão deste adicionando novos recursos e customizando o comportamento. Visando isso o lindbergframework fornece uma forma prática e simples de você criar suas próprias validações e adicionar ao contexto do framework, caso queria. Lembrando que para se criar um bean de validação basta implementar a interface IValidation mas se quiser fazer uso da classe ValidationFactory para obter instancias dos seus próprios beans de validação também é necessário estender o contexto de validações do framework.

Para estender o contexto é necessário criar os beans em um arquivo XML de contexto do spring inicialmente e esse arquivo deve estar dentro do classpath do projeto. Abaixo é mostrado um exemplo de um XML, o arquivo minhasValidacoesContext.xml, com a definição de beans validations externos ao lindbergframework.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

       xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

       xmlns:context=*"http://www.springframework.org/schema/context"*

       default-autowire=*"byName"*

       xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-2.5.xsd"*>

        <bean id=*"minhaValidation"* class=*"validacoes.MinhaValidation1"*></bean>

        <bean id=*"minhaValidation2"* class=*"validacoes.MinhaValidation2"*></bean>

        <bean id=*"minhaValidation3"* class=*"validacoes.MinhaValidation3"*></bean>

</beans>

No arquivo XML acima são adicionados 3 beans validations com os id´s (minhaValidation, minhaValidation2 e minhaValidation3). Isso indica que para obter instancias desses validations dentro do contexto do lindbergframework basta utilizar o método getValidationBean passando um desses ID´s definidos no arquivo XML que a instancia requerida será retornada, caso exista.

Com o XML acima definido agora é preciso estender realmente o contexto do framework. O contexto de validações do framework pode ser acessado através da classe *LindbergSpringValidationBeanFactory*. Da mesma forma para estender o contexto precisamos dessa classe. O método estático*initAdditionalContext(new String[])* desta classe é usado para obter uma instancia de *LindbergSpringValidationBeanFactory* com o contexto estendido adicionando os beans definidos no XML acima. Como o contexto de validações do lindbergframework é um singleton, ou seja é mantida apenas uma instancia deste em memória, então todas as classes que utilizam o contexto de validação dentro do projeto terá acesso automático a nova extensão do contexto. Abaixo é mostrado como executar as ações definidas acima. Vamos considerar que o XML de extensão do contexto definido acima esteja em*spring/conf.xml.*Observação: mesmo adicionando um contexto externo o contexto de validações padrão do lindbergframework é iniciado também. Por isso você pode definir sua própria fábrica de validações estendendo de ValidationFactory e adicionando métodos para obtenção de isntancias de suas próprias validações.

*\*  Estender o contexto*

LindbergSpringValidationsBeanFactory context = LindbergSpringValidationsBeanFactory

                                                          .*initAdditionalContext*(**new** String[] {"spring/conf.xml"});

*\*  Obter uma instancia de um bean de validação esteja ele no contexto interno ou externo do framework*

*IValidation validdationBean = ValidationFactory.getValidationBean(*“minhaValidation”*);*

**7. Como usar validações externas (criadas por você) para definir validações via annotations do lindbergframework**

A annotation @Valid que define uma validação específica para um campo específico de um bean tem um atributo *value* que recebe qual validation será utilizada. Este atributo recebe o ID do bean validation eu está no contexto de validação do lindbergframework ou o caminho completamente qualificado (nome da classe acompanhado do seu pacote completo) da classe responsável pela validação do campo. Essa classe deve implementar IValidation. Lembrando que para usar o ID da validation é necessário que este esteja dentro do contexto e quando o validation é definida externamente ao framework deve ser utilizado o mecanismo de extenção de contexto descrito no tópico anterior. Abaixo é mostrado o uso da annotation @Valid com o ID da validation NotNull dentro do framework e a mesam validação com o nome completamente qualificado.

*\*  Usando ID*

@Valid(value = "notNullValidation") ou @Valid("notNullValidation")

*\*  Usando nome completamente qualificado*

@Valid(value = "org.lindbergframework.validation.impl.NotNullValidation")

ou

@Valid("org.lindbergframework.validation.impl.NotNullValidation")

**8. Projeto Exemplo de uso do lindbergframework**

Acesse o item de menu donwloads para obter um projeto com os exemplos descritos acima.