SPRING - VALIDAÇÃO

Altere o arquivo **formulario.jsp** como segue:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<%@ page language="java" contentType="text/html; charset=UTF-8"
      pageEncoding="UTF-8"%>
<%@ page
      import="java.util.*, br.edu.ufabc.aulaspring.dao.*,
br.edu.ufabc.aulaspring.modelo.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
      <head>
            <title>Inserção no Cadastro</title>
      </head>
      <body>
            <form name="insereAluno" action="insere" method="POST">
            Nome:  <input  type="text"  id="nome"  name="nome"  /><form:errors
path="aluno.nome" cssStyle="color:red" /> <br />
            Email:  <input  type="text"  id="email"  name="email"  /><form:errors
path="aluno.email" cssStyle="color:red" /> <br />
            Endereço:   <input   type="text"   id="endereco"   name="endereco"
/><form:errors path="aluno.endereco" cssStyle="color:red" /> <br />
            <input type="submit" value="Inserir"/>
            </form>
      </body>
</html>
```

Observe que colocar a tag `<form: errors...>` na mesma linha do input (antes do `<br />`) facilita saber a qual campo a mensagem se refere.

No arquivo **AlunoController.java** modifique o código conforme destacado abaixo:

```
package br.edu.ufabc.aulaspring.controller;

import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import br.edu.ufabc.aulaspring.dao.AlunoDAO;
import br.edu.ufabc.aulaspring.modelo.Aluno;

@Controller
public class AlunoController {
	@RequestMapping("novoAluno")
	public String form() {
		return "formulario";
	}

	@RequestMapping("insere")
	public String insere(@Valid Aluno aluno, BindingResult result) {
		if (result.hasErrors()) {
			return "formulario";
		}
		AlunoDAO dao = new AlunoDAO();
		dao.insere(aluno);
		return "adicionado";
	}
}
```

No arquivo **Aluno.java** modifique o código conforme destacado abaixo:

```
package br.edu.ufabc.aulaspring.modelo;

import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class Aluno {
	private Long id;

	@NotNull @Size(min=5)
	private String nome;

	@NotNull @Size(min=5)
	private String email;

	@NotNull @Size(min=5)
	private String endereco;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getEndereco() {
		return endereco;
	}

	public void setEndereco(String endereco) {
		this.endereco = endereco;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}
}
```

Algumas observações sobre o arquivo Aluno.java

VALIDAÇÃO DOS CAMPOS:

Se quisermos utilizar apenas a anotação `@NotNull`, teremos um problema porque os input HTML vazios submetem uma string vazia e, portanto, passam pela validação. Assim, para contornar esse problema usamos a anotação `@Size` junto com `@NotNull`. Podemos ainda utilizar a anotação `@NotEmpty` que retorna a resposta "may not be empty" e o código fica assim:

```
public class Aluno {
	private Long id;

	@NotEmpty
	private String nome;

	@NotEmpty
	private String email;

	@NotEmpty
	private String endereco;
	.
	.
	.
```

VALIDAÇÃO DO EMAIL:

Se quisermos validar o campo e-mail checando se é um e-mail válido, podemos utilizar uma anotação que checa uma expressão regular. Veja o exemplo:

```
public class Aluno {
	private Long id;

	@NotEmpty
	private String nome;

	@Pattern(regexp="^[_A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$")
	private String email;

	@NotEmpty
	private String endereco;
	.
	.
	.
```

Expressões regulares são estudadas na disciplina de Linguagens Formais e Automata do curso de Ciência da Computação. Nossa validação checa se: (a) existe uma palavra ou então duas palavras separadas por ponto antes do símbolo @; (b) existe o símbolo @; (c) existe uma palavra depois do símbolo @. A expressão também permite duas ou três palavras separadas por pontos depois da @. Veja a explicação detalhada a seguir:

^	= início da linha
[_A-Za-z0-9-]+	= essa palavra deve ser formada pelos caracteres entre colchetes [ ] e deve ter um ou mais caracteres (+)
(\\.[_A-Za-z0-9-]+)*	= essa palavra é opcional, o que é indicado por ela estar entre parênteses e ter um símbolo * logo após o parênteses que fecha. Essa palavra deve começar por um ponto "." seguida de um ou mais dos caracteres entre colchetes [ ]

@	= deve ter um símbolo "@"
[A-Za-z0-9]+	= essa palavra deve ser formada pelos caracteres entre os colchetes [ ] e deve ter um ou mais caracteres (+)
(\\.[A-Za-z0-9]+)*	= essa palavra é opcional, o que é indicado por ela estar entre parênteses e ter um símbolo * logo após o parênteses que fecha. Essa palavra deve começar por um ponto "." seguida de um ou mais dos caracteres entre colchetes [ ]
(\\.[A-Za-z]{2,})	= essa palavra começa com um ponto "." e deve ser formada pelos caracteres entre os colchetes [ ], com comprimento mínimo de 2
$	= fim da linha