02

Validação de dados

Agora que já conseguimos adicionar e listar nossas contas, e o básico já está funcionando, precisamos trabalhar um pouco melhor no formulário. Veja, por exemplo, que conseguimos adicionar uma conta sem descrição! Isso não deveria acontecer, afinal, toda conta precisa ser descrita.

Como primeira alternativa, podemos fazer a validação no próprio controller. Por exemplo, antes de adicionar no banco, poderíamos verificar se a descrição está vazia e, caso isso seja verdade, redirecionar novamente para o formulário:

```
@Controller
public class ContaController {
  @RequestMapping("/adicionaConta")
  public String adiciona(Conta conta) {
    // validando a descrição
    if(conta.getDescricao() == null || conta.getDescricao().equals("")) {
      return "formulario";
    }

    ContaDAO dao = new ContaDAO();
    dao.adiciona(conta);
    return "conta-adicionada";
  }
}
```

Essa é uma alternativa válida. O problema dela é que se quisermos validar outros 10 campos, teríamos um conjunto de `if` s muito grande, de difícil manutenção. Cuidar da qualidade do código é sempre fundamental.

Uma alternativa para isso é usar a especificação de Bean Validation, que existe desde o Java EE 6. Com ela, nós podemos simplesmente anotar os atributos da classe com anotações específicas de validação, que o Spring entenderá e fará a validação para nós. Por exemplo, se quisermos que o atributo Descrição seja obrigatório e deve ter no mínimo 5 caracteres, poderíamos anotá-lo da seguinte forma:

```
public class Conta {
  @NotNull @Size(min=5)
  private String descricao;

  // outros atributos e o resto da classe
}
```

Pronto! Basta agora pedirmos ao Spring que valide essa Conta, usando as anotações que colocamos. Isso também é bem simples, basta usarmos a anotação `@Valid` do próprio Spring, antes do parâmetro no método da nossa Action:

```
@Controller
public class ContaController {
  @RequestMapping("/adicionaConta")
  public String adiciona(@Valid Conta conta) {
    ContaDAO dao = new ContaDAO();
    dao.adiciona(conta);
    return "conta/conta-adicionada";
```

```
  }
}
```

Se testarmos agora, você verá que se deixarmos o campo Descrição em branco ou preenchê-lo com menos de 5 caracteres, uma exceção será lançada. Já está melhor, pois o Spring está fazendo essa validação. Mas o problema é que não queremos que a exceção seja exibida pro usuário; precisamos tratá-la, e por exemplo, avisar o usuário sobre o que aconteceu.

Para isso, precisamos receber mais um parâmetro na nossa Action, do tipo `BindingResult`. É nesse parâmetro que o Spring nos avisará se houve um erro de validação, e nos deixará livres para tratarmos do jeito que quisermos. Veja o código abaixo, onde perguntamos pra esse novo objeto se existe algum erro de validação, através do método `hasErrors()`:

```
@Controller
public class ContaController {
  @RequestMapping("/adicionaConta")
  public String adiciona(@Valid Conta conta, BindingResult result) {
    // se tiver erro, redirecione para o formulário
    if(result.hasErrors()) {
      return "conta/formulario";
    }

    ContaDAO dao = new ContaDAO();
    dao.adiciona(conta);
    return "conta/conta-adicionada";
  }
}
```

Falta pouco agora, pois nossa aplicação já devolve o usuário para a tela de formulário, caso alguma validação falhe. De volta ao formulário, precisamos agora exibir a mensagem amigável, pois nesse momento, nada aparece.

Para isso, faremos uso de uma taglib do Spring, que ajuda a exibir a mensagem de erro. Essa taglib chama-se `form:errors`. E passamos pra ela o "caminho" do atributo que queremos exibir o erro. Por exemplo, se quiséssemos exibir o erro do conta.descricao, faríamos:

```
<form:errors path="conta.descricao" />
```

Vamos colocar essa mensagem agora dentro da nossa JSP:

```
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

    <form action="<c:url value='/conta/adicionaConta'/>" method="post">
        Descrição: <br/>
        <textarea name="descricao" rows="5" cols="100"></textarea>
                <form:errors path="conta.descricao" />
        <br/>
        Tipo: <br/>
        <select name="conta.tipo">
            <option value="ENTRADA">Entrada</option>
            <option value="SAIDA">Saída</option>
        </select>
```

```
        <br/><br/>
        <input type="submit" value="Adicionar"/>
    </form>
```

Precisamos agora só customizar a mensagem de erro, pois por enquanto o Spring nos devolve uma genérica. Para isso, precisamos criar um arquivo com a extensão .properties, dentro da pasta /src. Chamamos esse arquivo, obrigatoriamente, de `ValidationMessages.properties` . Dentro dele, colocaremos o seguinte conteúdo:

```
conta.formulario.sucesso=Conta adicionada com sucesso.
conta.formulario.descricao.obrigatoria=Descrição é obrigatória.
conta.formulario.descricao.tamanho=Descrição deve ter o mínimo de {min} caracteres.
```

Veja que escrevemos um conjunto de chaves e valores. No fundo, são as mensagens que queremos exibir na tela. Na classe `Conta` novamente, vamos apontar cada erro de validação para sua respectiva mensagem:

```
class Conta {
  @NotNull(message="{conta.formulario.descricao.obrigatoria}")
  @Size(min=5, message="{conta.formulario.descricao.tamanho}")
  private String descricao;

  // classe continua aqui
}
```

Veja que passamos a chave da mensagem que está no arquivo `ValidationMessages.properties` para a anotação. Agora o Spring saberá qual mensagem pegar em caso de erro de validação!

Pronto! Nossa validação agora funciona da maneira que esperamos: se ela falhar, o Spring redireciona para o formulário e mostra uma mensagem amigável em português; se não falhar, ele então adiciona a Conta e a salva no banco de dados.

Validar os dados recebidos pelo usuário é fundamental. Lembre-se que ele pode digitar qualquer coisa ali. É sua tarefa fazer a validação. O Spring junto com a especificação de Bean Validation ajuda muito nessa tarefa!