



Você está em

DevMedia

Artigo

Hibernate Validator: como validar objetos e ter mais controle sobre os dados

Aprenda neste artigo como melhorar a qualidade dos dados que sua aplicação necessita para um correto funcionamento.

Por que eu devo ler este artigo: Este artigo é útil por apresentar como expressar e validar regras existentes no domínio de negócio de uma aplicação. Para isso, aprenderemos de forma básica e simples os:

[Ver mais](#)

Marcar como concluído



Anotar

Artigos



Java



Hibernate Validator: como validar objetos e ter mais controle sobre os dados



5





Você está em

DevMedia

campo que receberá uma data não especifique o formato esperado pela aplicação, não há como garantir que o usuário entrará com o dado no formato desejado, o que acaba se tornando um desafio aos programadores.

Diante disso, é necessário realizar o tratamento adequado de todas as informações que são enviadas para evitar que problemas inesperados venham a ocorrer durante o processamento, como exceções que interromperão a execução do sistema, problemas de integridade ou corrupção dos dados e também questões relacionadas à falha de segurança.

Cada informação passada para a aplicação possui um propósito. Assim, é natural que algumas regras sejam introduzidas com o intuito de controlar a interação entre o usuário e o sistema, garantindo deste modo a correta execução das regras de negócio. E com o intuito de garantir o cumprimento dessas regras que o conceito de validação foi implementado. Validar é determinar se os dados capturados pelo sistema seguem as regras lógicas definidas para manter a sua consistência.

Neste contexto, conseguir consolidar e implementar a validação usando uma solução de qualidade como o Hibernate Validator pode melhorar significativamente a confiabilidade do software, especialmente ao longo do tempo, e torná-lo mais amigável ao usuário.

Considerando que as políticas de validação não estruturadas e não controladas vão levar ao aumento dos custos de suporte e manutenção, uma estratégia de validação consolidada pode minimizar significativamente o efeito cascata de



5





Você está em

DevMedia

pela empresa Red Hat que permite a validação dos dados, presentes nas classes que modelam o domínio da aplicação, em qualquer arquitetura (Web, Desktop, etc.) e em tempo de execução. A motivação para a utilização dessa biblioteca é poder validar os dados diretamente no domínio da aplicação, em vez de realizar esse processo por camadas. Dessa forma, é possível validar campos numéricos, definir se datas informadas serão obrigatoriamente maiores ou menores que a data atual, verificar se o campo pode ser vazio ou não diretamente nas classes de domínio de maneira centralizada e flexível, mantendo o código claro e enxuto.

A partir disso, ao longo desse artigo abordaremos os principais conceitos e características do Hibernate Validator, e veremos como desenvolver um sistema de cadastro de projetos considerando como um dos seus principais requisitos não funcionais a confiabilidade do sistema. Para tanto, serão empregados, além do Validator, a linguagem de programação Java, o ambiente de desenvolvimento Eclipse integrado ao Maven, o sistema de gerenciamento de banco de dados MySQL, o *framework* JSF e o container web Tomcat. Ademais, o Hibernate será usado como solução integrante da camada de persistência, viabilizando a interface entre a aplicação e o MySQL.

Hibernate Validator

O Hibernate é um framework de mapeamento objeto relacional muito popular entre os desenvolvedores Java. Distribuído sob a licença LGPL, foi criado por Gavin King em 2001, sendo atualmente o framework de persistência de dados

mais utilizado. Segundo a documentação oficial: “o Hibernate suporta a maioria das



5





Você está em

DevMedia

SQL.

O Hibernate Validator, por sua vez, é a implementação de referência da *JSR 303 – Bean Validation API*. Disponibilizada em dezembro de 2009, a partir da especificação do Java EE 6, na qual foi introduzida a especificação Bean Validation 1.0, o objetivo principal dessa API é permitir a validação dos dados de forma fácil e rápida, através do uso de anotações e, de forma alternativa, utilizando arquivos XML na configuração.

Com o lançamento mais recente da plataforma Java EE, agora na versão 7, a JSR 349 foi divulgada, introduzindo a versão 1.1 da API de validação e trazendo novidades como:

- Uso de injeção de dependências e integração com CDI;
- Validação de parâmetros e retornos de métodos;
- Uso de grupos de conversão;
- Suporte à concatenação de mensagens de violação através de *expression language*;
- Integração com outras especificações, como JAX-RS.

Antes do surgimento dessa API, cada framework implementava um mecanismo proprietário para validar as informações, o que criava problemas de incompatibilidade e dificultava a integração com outros frameworks.

Com o surgimento dessa especificação, possibilitou-se uma API padrão para validação que é flexível o suficiente para ser utilizada pelos mais diversos tipos de frameworks. Além disso, a Bean Validation API viabiliza a validação de dados nas



5





Você está em

DevMedia

centralizado, uma vez que os objetos destas classes normalmente trafegam entre as camadas da aplicação.

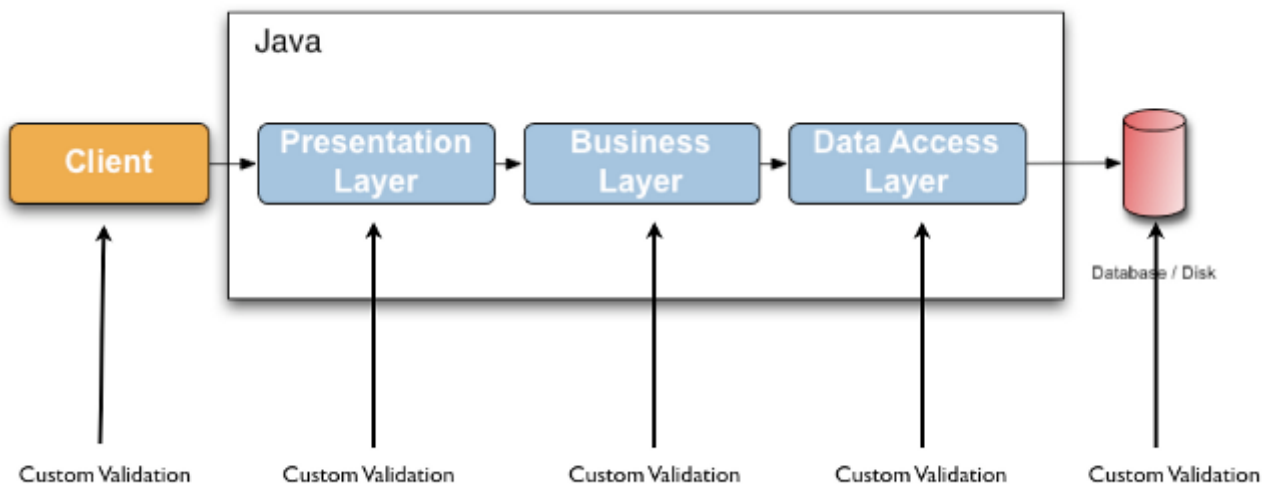


Figura 1. Validação realizada em várias camadas. Fonte: Hibernate Validator Reference Guide.

O Hibernate Validator, como principal implementação da Bean Validation API, segue a premissa estabelecida pelo DRY (*Don't Repeat Yourself*), que especifica uma forma de adicionar regras e respectivas verificações para validação automática dos dados, de maneira que estas validações sejam implementadas uma e somente uma vez em toda a aplicação e gerenciadas de maneira centralizada, eliminando a duplicação entre as diversas camadas. A **Figura 2** mostra esta configuração, em que todas as camadas podem invocar a verificação concentrada em um único lugar. Com isso, evita-se a reescrita de código, uma vez



5





Você está em

DevMedia

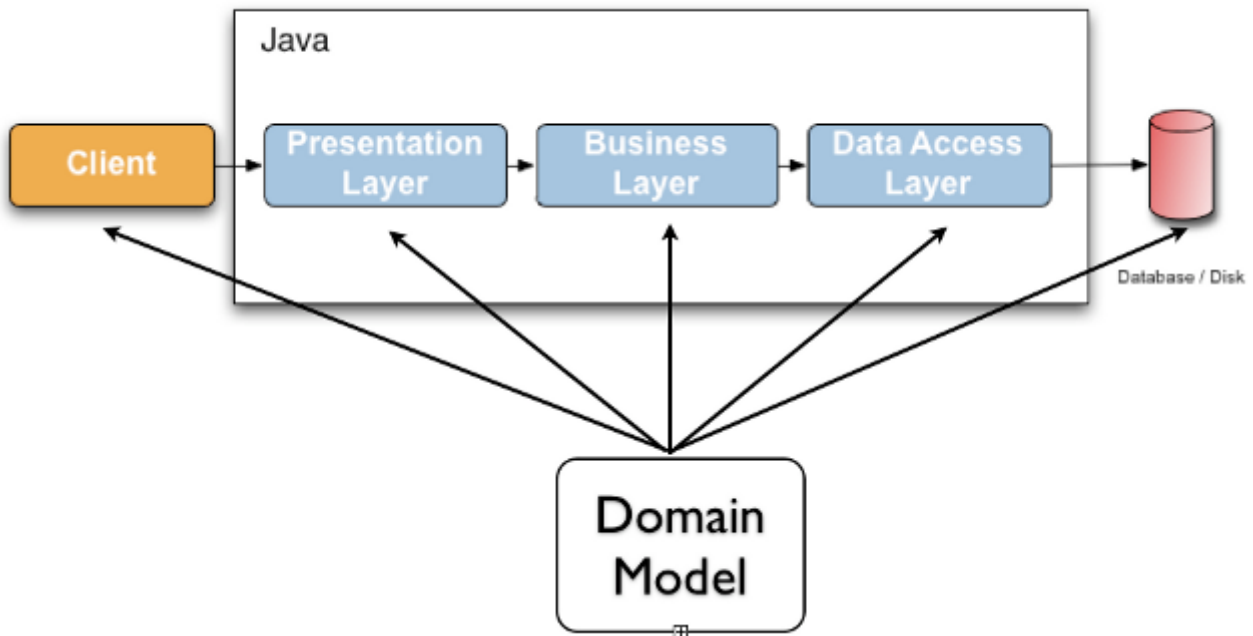


Figura 2. Validação centralizada no modelo de domínio. Fonte: Hibernate Validator Reference Guide.

As validações do framework são definidas através de restrições realizadas diretamente nos Java Beans, especificados no modelo de domínio. Essas restrições são utilizadas para definir regras a respeito dos dados que podem ser atribuídos a um objeto. Assim, quando um processo de validação de dados é executado, é feita uma verificação para checar se os mesmos estão de acordo com as regras estabelecidas.

As restrições são demarcadas através de Java *annotations*, sendo possível adicioná-las tanto nas propriedades do bean quanto nas chamadas de métodos, garantindo que os retornos dos métodos ou os valores dos parâmetros sejam validados. Outra maneira de adicionar regras é diretamente na classe. Neste caso,



5





Você está em

DevMedia

disponíveis. Dessa forma, o validador da restrição tem acesso ao objeto **Carro**, possibilitando comparar o número de assentos com o número de passageiros.

O Hibernate Validator oferece várias validações, mas não limita o desenvolvedor a elas, ou seja, também possibilita a criação de novas regras, uma vez que nem sempre esses validadores serão suficientes. Por exemplo, a biblioteca dispõe de anotações que verificam a numeração de cartões de crédito, e-mail, URL e, até mesmo, CNPJ, CPF e Título Eleitoral, mas também podem ser criadas novas validações como, por exemplo, para certificar se o valor inserido em um campo é um valor válido para a placa de um carro.

As restrições padrão são:

- **@NotNull**: Verifica se um dado não é nulo;
- **@Null**: Verifica se um dado é nulo;
- **@AssertFalse** e **@AssertTrue**: Checa se o dado é verdadeiro ou falso. Estas validações podem ser aplicadas ao tipo primitivo **boolean** ou à classe **Boolean**;
- **@Min** e **@Max**: Validam o valor mínimo ou o valor máximo para os tipos **BigDecimal**, **BigInteger**, **String**, **byte**, **short** e suas classes wrappers correspondentes;
- **@Size**: Valida se o tamanho do dado está entre os valores especificados nos atributos **min** e **max**. Aplica-se a Strings, Collections e Arrays;
- **@Pattern**: Valida o dado de acordo com uma expressão regular especificada pelo atributo **regexp** da anotação. Funciona somente para dados do tipo **String**;



5





Você está em

DevMedia

- **@Valid:** Opção utilizada para validar atributos que referenciam outras classes, impondo uma validação recursiva aos objetos associados.

Cada *annotation* é associada a uma implementação de validação, que verifica se a instância da entidade obedece à regra relacionada. O Hibernate faz isto automaticamente antes que uma inserção ou atualização seja realizada no banco de dados, mas o desenvolvedor também pode chamar a validação a qualquer momento em sua aplicação.

Nota: Todos os elementos do Hibernate Validator (classes, interfaces, annotations, etc.) pertencem ao pacote **javax.validation**. Portanto, sempre que algum elemento da API for referenciado, será necessário realizar o import desse pacote ou mesmo de seus subpacotes.

Configuração do exemplo

Vamos partir para a parte prática e desenvolver uma aplicação que possibilite ao leitor visualizar como o *framework* funciona e sua utilidade.

No entanto, antes de começar a codificar, é importante instalar os softwares que nos auxiliarão nesse trabalho e também preparar o ambiente de desenvolvimento para uso do Validator.

Preparando o ambiente de desenvolvimento



5





Você está em

DevMedia

ter um JDK instalado no sistema. Na seção **Links** está disponível o endereço para download do JDK e da IDE, cuja versão adotada no exemplo foi o Eclipse Luna SR2 (4.4.2).

Concluído o download, descompacte-o no seu sistema de arquivos, em um local de sua preferência. Neste artigo optamos pela pasta `C:\Eclipse_Luna`. Em seguida, é preciso apenas executar o arquivo `eclipse.exe`.

Integrando o Maven ao Eclipse

O Maven é uma das ferramentas mais conhecidas e utilizadas por profissionais que adotam o Java em seus projetos. Com o objetivo de simplificar o gerenciamento de dependências e o ciclo de vida do desenvolvimento (compilação, testes unitários, empacotamento e distribuição), esta solução da Apache garante as seguintes metas:

- Prover um padrão para o desenvolvimento de aplicações;
- Fornecer mecanismos para uma clara definição da estrutura do projeto;
- Controlar versão e artefatos;
- Viabilizar/facilitar o compartilhamento de bibliotecas entre projetos.

Como maior atrativo, a principal facilidade viabilizada pelo Maven é o gerenciamento de dependências e este será o nosso motivador para empregá-lo em parceria com o Eclipse.

O endereço para download desta solução encontra-se na seção **Links**. Ao acessá-



5





Você está em

DevMedia

plugin e uma instalação interna do Maven, não precisaremos adicionar o M2E.

Para conhecer a versão do Maven que está configurada, com o Eclipse aberto, acesse o menu *Window > Preferences* e escolha a opção *Maven > Installations*, como sugere a **Figura 3**. Ao fazer isso você poderá notar a presença da versão “embarcada”. Apesar disso, vamos optar pela versão do Maven que baixamos anteriormente, por ser mais recente.



5





Você está em

DevMedia

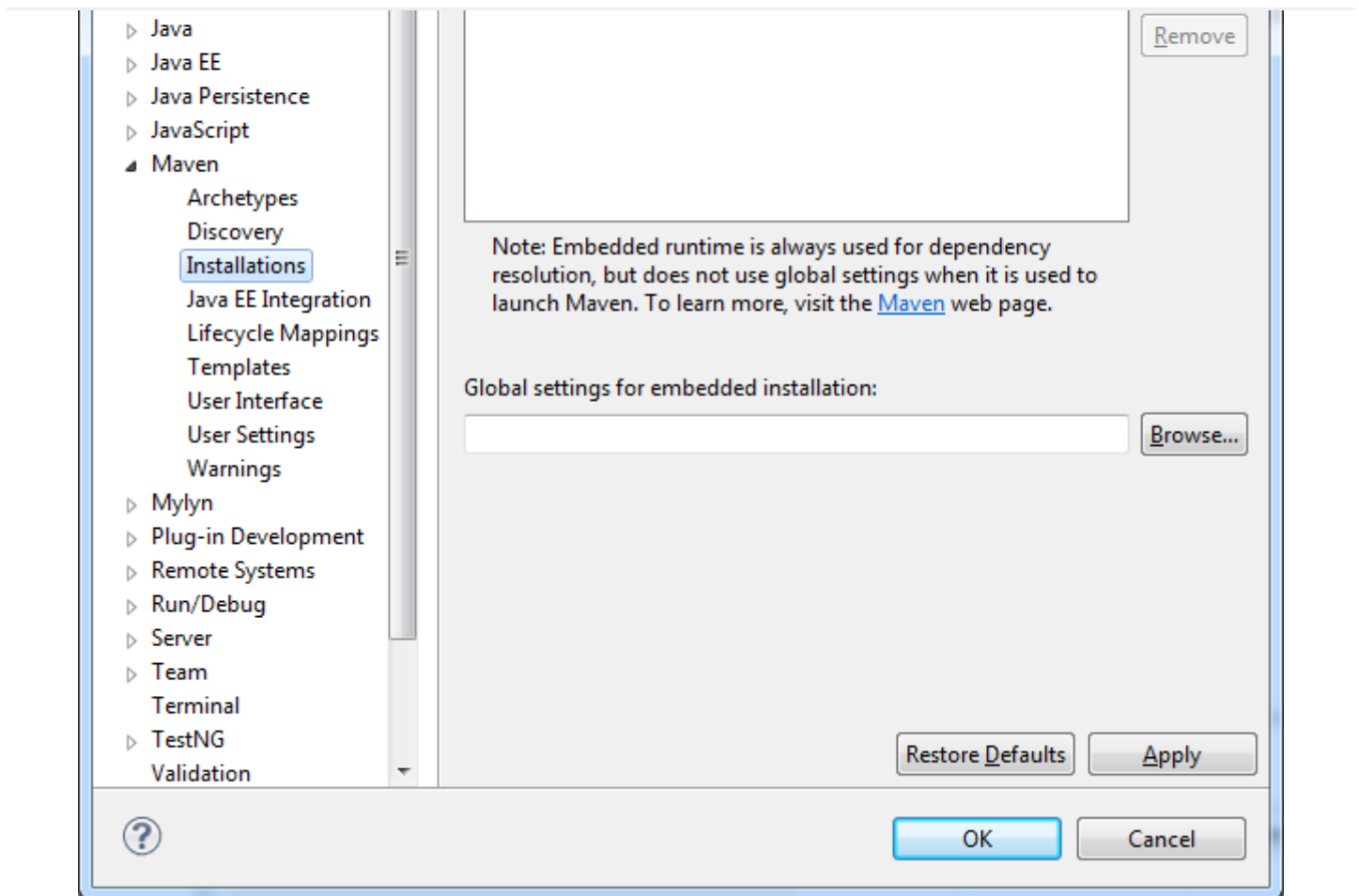


Figura 3. Instalação padrão do Maven no Eclipse.

Deste modo, para adicionar o Maven ao Eclipse, ainda na **Figura 3**, clique em *Add* e selecione a pasta *C:\Maven\apache-maven*. Feito isso, automaticamente o arquivo de configuração do Maven é localizado e a versão desejada é adicionada à IDE. Para concluir, clique em *Ok*.

Adicionando o Tomcat ao Eclipse



5





Você está em

DevMedia

utilizada do Java seja igual ou superior a 7.

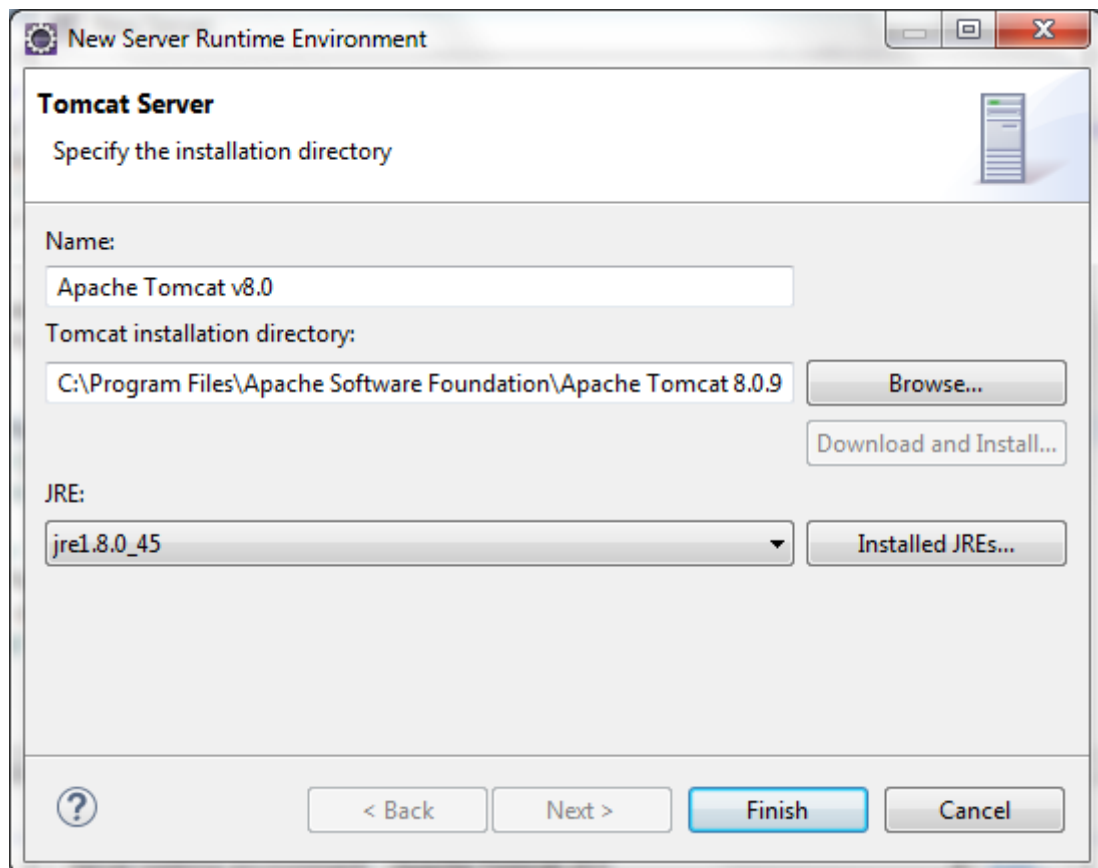


Figura 4. Adicionando o Tomcat ao Eclipse.

O sistema gerenciador de banco de dados

O sistema gerenciador de banco de dados que utilizaremos para viabilizar a persistência em nosso exemplo será o MySQL (veja o endereço na seção **Links**). Ao se deparar com as diferentes versões, opte pela adotada neste material, a *Community Server 5.6.22*.



5





Você está em

DevMedia

3. Chegou o momento de especificar o tipo de instalação e depois clicar em *Next*. A escolha vai depender de cada profissional e para que ele vai usar o MySQL. Aqui, optamos pela opção *Custom*;
4. No próximo passo o usuário deve escolher os produtos que deseja instalar. Escolha o *MySQL Server 5.6.22* e clique em *Next*. Feito isso, serão exibidos os produtos selecionados na etapa anterior. Clique então em *Execute* para continuar;
5. No final do processo de instalação, a coluna *Status* passará a exibir a informação “*Complete*”. Assim, pressione *Next* mais duas vezes;
6. Agora o usuário deve escolher o tipo de configuração. Para os nossos objetivos, apenas mantenha os valores padrão e clique em *Next*;
7. Na próxima tela, informe a senha do administrador e continue (*Next*);
8. Chega o momento de configurar o MySQL como um serviço do Windows. Para tanto, apenas mantenha os valores *default* e clique em *Next*;
9. A janela apresentada mostra todas as configurações que serão aplicadas. Confirme as escolhas feitas previamente, clique em *Execute* e, na tela seguinte, em *Next*;
10. Por fim, uma janela confirma a configuração do MySQL. Então, pela última vez, clique em *Next*, e logo depois, em *Finish*.

Desenvolvendo o cadastro de projetos

Agora que temos o banco de dados instalado, assim como o Eclipse e o Maven integrados e o Tomcat incorporado ao Eclipse, vamos partir para a construção de



5





Você está em

DevMedia

operações necessárias para se ter um cadastro: salvar, atualizar, listar e excluir.

Além disso, cada projeto conterà as seguintes propriedades: código, nome, nome do responsável pelo projeto, CPF do responsável, e-mail, data de início, data de fim e descrição. A **Figura 5** mostra a representação gráfica da tabela.

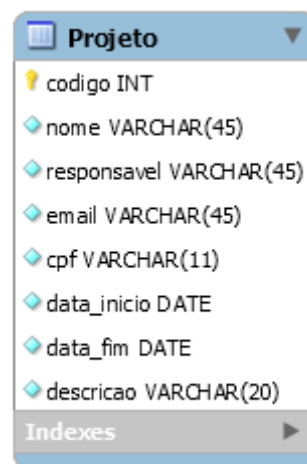


Figura 5. Representação gráfica da tabela projeto.

Criando o banco de dados

Com o MySQL instalado, crie o banco *projetobd* executando o script SQL indicado na **Listagem 1**. Neste comando, as duas primeiras linhas especificam a geração do banco de dados e como entrar em seu contexto. A tabela e seus respectivos atributos são especificados na sequência do código.



5





Você está em

DevMedia

```
4 CREATE TABLE IF NOT EXISTS projeto (  
5     codigo INT NOT NULL,  
6     nome VARCHAR(45) NOT NULL,  
7     responsavel VARCHAR(45) NOT NULL,  
8     email VARCHAR(45),  
9     cpf VARCHAR(14) NOT NULL,  
10    data_inicio DATE,  
11    data_fim DATE,  
12    descricao VARCHAR(20),  
13    PRIMARY KEY (codigo))  
14 ENGINE = InnoDB;
```

Criando o projeto Maven no Eclipse

Com todos os recursos instalados, vamos partir para a elaboração do projeto web criando um novo com o auxílio do Maven, no Eclipse. Como utilizaremos anotações JPA, é importante que o projeto adote o Java 1.5+.

Dito isso, com o Eclipse aberto, clique em *New > Other*. Na tela que surgir, selecione *Maven > Maven Project*, marque a opção *Create a simple project (skip archetype selection)* e pressione *Next*.

Na próxima tela, devemos identificar o projeto, o que é feito ao especificar estas opções e clicar em *Finish*:

- Group ID: **br.com.devmedia;**
- Artifact ID: **hibernate-validator;**
- Packaging: **war;**



5





Você está em

DevMedia

configurações relacionadas às versões do Java, JSF e do módulo web. Portanto, selecione a versão 3.1 para *Dynamic Web Module*; na opção *Java*, escolha a versão 1.8; e para o JavaServer Faces opte pela versão 2.2.

Apesar desses ajustes, observe que a estrutura do projeto ainda não está definida com as configurações de um projeto Maven. Para isso, é necessário atualizá-lo. Portanto, clique com o botão direito sobre o projeto e acesse o menu *Maven > Update Project*. Na tela que aparecer, selecione a opção *hibernate-validator*, que acabamos de criar, e clique em *Ok* para atualizá-lo de acordo com as definições do Maven. Assim, o projeto passará a ter a estrutura apresentada na **Figura 6**.

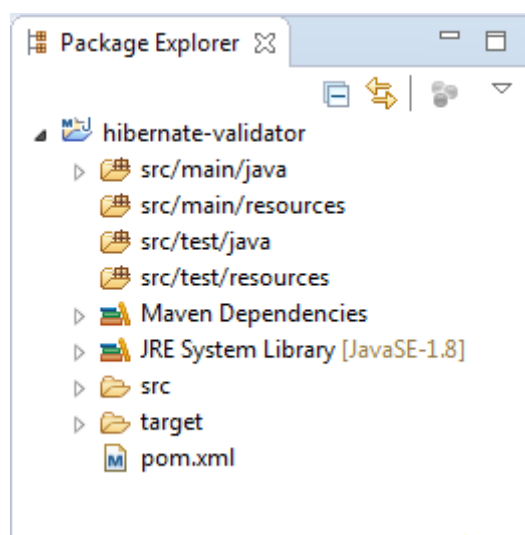


Figura 6. Visão Package Explorer da estrutura de diretórios do projeto.

Adicionando as dependências



5





Você está em

DevMedia

Para inserir as dependências, clique com o botão direito sobre o POM e acesse *Maven > Add Dependency*. Na nova janela, informe o nome das bibliotecas que deseja utilizar (Hibernate Core, Hibernate Validator, JSF e o driver do MySQL), como exemplificado na **Figura 7**, e clique em *Ok*.

Concluída essa etapa, o *pom.xml* terá o conteúdo mostrado na **Listagem 2**.

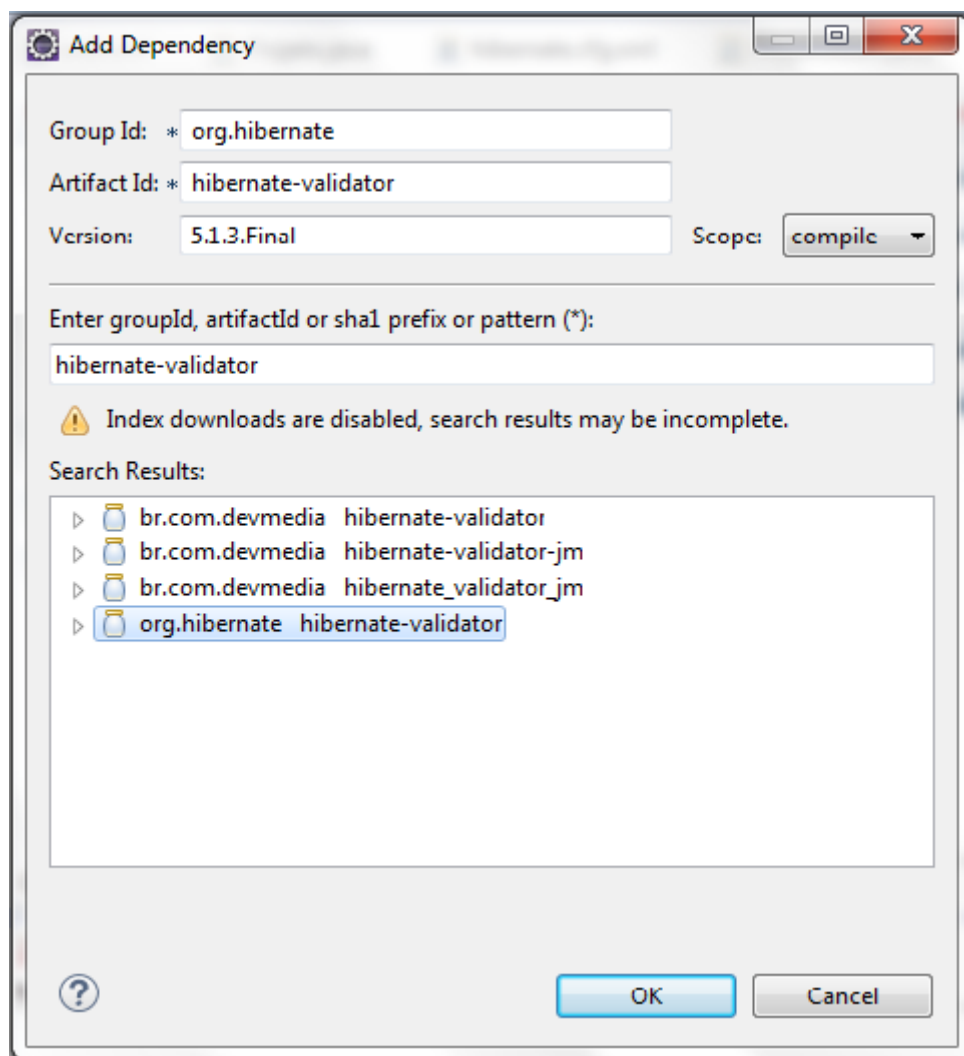


Figura 7. Adicionando dependências no arquivo pom.xml.



5





Você está em

DevMedia

```
4 http://maven.apache.org/xsd/maven-4.0.0.xsd >
5 <modelVersion>4.0.0</modelVersion>
6 <groupId>br.com.devmedia</groupId>
7 <artifactId>hibernate-validator</artifactId>
8 <version>0.0.1-SNAPSHOT</version>
9 <packaging>war</packaging>
10 <dependencies>
11   <dependency>
12     <groupId>javax</groupId>
13     <artifactId>javaee-api</artifactId>
14     <version>7.0</version>
15   </dependency>
16   <dependency>
17     <groupId>javax.servlet</groupId>
18     <artifactId>javax.servlet-api</artifactId>
19     <version>3.1.0</version>
20     <scope>provided</scope>
21   </dependency>
22   <dependency>
23     <groupId>org.hibernate</groupId>
24     <artifactId>hibernate-core</artifactId>
25     <version>4.3.10.Final</version>
26   </dependency>
27   <dependency>
28     <groupId>com.sun.faces</groupId>
29     <artifactId>jsf-api</artifactId>
30     <version>2.2.4</version>
31   </dependency>
32   <dependency>
33     <groupId>com.sun.faces</groupId>
34     <artifactId>jsf-impl</artifactId>
35     <version>2.2.9</version>
36   </dependency>
37   <dependency>
38     <groupId>mysql</groupId>
39     <artifactId>mysql-connector-java</artifactId>
```



5





Você está em

DevMedia

```
47     </dependencies>
48     <build>
49         <plugins>
50             <plugin>
51                 <groupId>org.apache.maven.plugins</groupId>
52                 <artifactId>maven-compiler-plugin</artifactId>
53                 <version>3.1</version>
54                 <configuration>
55                     <source>1.8</source>
56                     <target>1.8</target>
57                 </configuration>
58             </plugin>
59         </plugins>
60     </build>
61 </project>
```

Selecionadas as dependências, após clicar com o botão direito do mouse sobre o projeto, acesse *Maven > Update Project* e automaticamente as bibliotecas serão incorporadas ao *classpath* do projeto.

Criando o modelo e inserindo as anotações de validação

Nosso próximo passo é criar e mapear a entidade *projeto*, assim como definir as restrições que desejamos. Deste modo, crie a classe **Projeto**, classe de negócios que será mapeada pelo Hibernate como uma tabela no banco de dados. Ademais, como será possível notar em seu código, **Projeto** é um tipo de classe simples, definida como um POJO, que contém todos os atributos encapsulados através de



5





Você está em

DevMedia

que surgir, informe “Projeto” no campo nome. A **Listagem 3** mostra o código dessa classe.

Listagem 3. Código da classe Projeto.

```
1 package br.com.jm.projeto.model;
2
3 //imports omitidos
4
5 @Entity
6 @Table
7 public class Projeto implements Serializable {
8
9     private static final long serialVersionUID = 1L;
10
11     public Projeto() {}
12
13     @Id
14     @Column
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     private int codigo;
17
18     @NotEmpty (message="Informe o nome do Projeto")
19     @Column
20     private String nome;
21
22     @NotEmpty (message="Informe o nome do Responsável pelo Projeto")
23     @Column
24     private String responsavel;
25
26     @CPF(message="CPF inválido ")
27     @NotEmpty(message="Informe o número do CPF")
```



5





Você está em

DevMedia

```
36     @Column
37     private String email;
38
39     @Future(message="A data deve ser maior do que a atual. Você digitou")
40     @Temporal(TemporalType.DATE)
41     @Column
42     private Date dataInicio;
43
44     @Temporal(TemporalType.DATE)
45     @Column
46     private Date dataFim;
47
48     @Future(message="A data deve ser maior do que a atual. Você digitou")
49     public Date getDataFim() {
50         return dataFim;
51     }
52
53     public void setDataFim(Date dataFim) {
54         this.dataFim = dataFim;
55     }
56
57     public boolean isFimdeSemana(@NotEmpty(message = "Data de término obrigatório"))
58         return true;
59     }
60     //gets e sets omitidos
61 }
```

O leitor mais atento identificará em **Projeto** anotações referentes ao Hibernate e ao Hibernate Validator. Logo no início, na linha 5, declaramos **@Entity**. Esta anotação sinaliza que haverá uma tabela relacionada a essa classe no banco de dados e que os objetos desta classe serão persistidos. Já as anotações **@Column** e



5





Você está em

DevMedia

@GeneratedValue, que geralmente acompanha **@Id** e que serve para indicar que o valor do atributo que define a chave primária deve ser criado pelo banco ao persistir o registro. E a anotação **@Temporal(TemporalType.DATE)**, presente nas **linhas 40 e 44**, especifica que os campos **dataInicio** e **dataFim** irão trabalhar com valores no formato de uma data.

As demais anotações são referentes ao processo de validação e serão explicadas a seguir:

- **Linhas 18, 22 e 27:** A anotação **@NotEmpty** é direcionada para atributos do String, Collection, Map ou Array e verifica se qualquer um desses não é nulo e nem vazio;
- **Linha 26:** A anotação **@Cpf**, inserida na versão 5.0.0 Alpha1 do Hibernate Validator, verifica se o valor informado corresponde a um CPF válido, de acordo com as regras nacionais;
- **Linha 31:** A anotação **@Size** validará a **String** para verificar se seu tamanho tem no máximo 10 caracteres. Observe neste caso que o atributo **message**, que personaliza a mensagem de erro a ser exibida, possui a expressão **max** entre chaves. Este é um recurso da especificação que permite inserir na mensagem de erro os valores dos próprios atributos das anotações. Além disso, concatenamos a mensagem de erro com a variável **validateValue**, recuperada via *Expression Language*, para exibir o valor digitado pelo usuário. A concatenação com *EL* é um novo recurso implementado na versão 1.1 de Bean Validation;
- **Linha 35:** A anotação **@Pattern** é utilizada quando se deseja checar um



5





Você está em

DevMedia

à data atual.

Vale ressaltar que a validação também pode ser realizada em métodos de acesso. A **linha 48** ilustra essa situação, onde a validação é feita diretamente no método **getDataFim()**.

Outro recurso disponível é a validação de parâmetros passados para os métodos. Para demonstrar esse recurso, implementamos o método de negócio **isFimdeSeamana()** na classe **Projeto** para verificar se a data cadastrada como término do projeto refere-se a um dia de final de semana. Na assinatura desse método, observe que antes do parâmetro **dataTermino**, colocamos uma validação especificando que o valor não pode ser vazio e uma mensagem de erro.

Criando o Controller da aplicação

A camada de controle do JSF é composta pelos Managed Beans, elementos responsáveis por controlar o fluxo de processamento e estabelecer a ligação entre nosso modelo e a camada de visão. Sabendo disso, para criar o nosso bean gerenciado, clique com o botão direito do mouse sobre o projeto, selecione *New > Class* e nomeie a classe como **ProjetoBean**. O seu código é apresentado na **Listagem 4**, e como verificado, deve ficar no pacote **br.com.jm.projeto.managedbean**.

Listagem 4. Código da classe ProjetoBean.



5





Você está em

DevMedia

```
0  @RequestScoped
7  public class ProjetoBean implements Serializable{
8
9      private static final long serialVersionUID = 1L;
10     private Projeto projeto = new Projeto();
11
12     public String salvar(){
13         ProjetoDao dao = new ProjetoDaoImp();
14         dao.save(projeto);
15         FacesContext.getCurrentInstance().addMessage(
16             null,
17             new FacesMessage(FacesMessage.SEVERITY_INFO,
18                 "Manutenção de projeto: ",
19                 "projeto incluído com sucesso!"));
20         return "sucesso";
21     }
22     //métodos get e set omitidos
23 }
```

Na **linha 6**, com a anotação **@RequestScoped**, definimos que esse *Managed Bean* terá escopo de requisição. Na linha 12, declaramos uma variável de instância do tipo **Projeto** e na **linha 14** implementamos o método **salvar()**, que realiza a validação do objeto **Projeto** e direciona o fluxo da aplicação para alguma página – neste caso, a página *sucesso.xhtml*, que criaremos em breve.

Criando a camada View da aplicação

O próximo passo será criar a página web por onde serão especificadas as informações do projeto. Essa página se comunica diretamente com os métodos e



5





Você está em

DevMedia

Neste arquivo podemos ver um formulário simples contendo campos de texto e um botão que chama o método `salvar()` do *Managed Bean* (**linha 28**). Na **linha 7**, observe a presença da tag `<f:validateBean>`. Esta tem como objetivo integrar o *Bean Validation* ao JSF. Assim, sinalizamos que todos os campos que estiverem dentro dela serão checados. Outra tag importante é a `<h:messages>`, vista na **linha 5**, que é responsável por mostrar na tela os possíveis erros que ocorrerão na validação.

Listagem 5. Formulário de entrada de dados.

```
1  <h:body>
2  <h2>Preencha o formulário abaixo</h2>
3  <f:view>
4  <h:form id="frmProjeto" method="post">
5  <h:messages style="color:red;margin:8px;" />
6  <h:panelGrid columns="2" style="horizontal-align:center">
7  <f:validateBean>
8  <h:outputText value="Nome do Projeto:" />
9  <h:inputText value="#{projetoBean.projeto.nome}" />
10 <h:outputText value="Responsável:" />
11 <h:inputText value="#{projetoBean.projeto.responsavel}" />
12 <h:outputText value="CPF do Responsável:" />
13 <h:inputText value="#{projetoBean.projeto.cpf}" />
14 <h:outputText value="Email do Responsável:" />
15 <h:inputText value="#{projetoBean.projeto.email}" />
16 <h:outputText value="Data de Inicio do Projeto:" />
17 <h:inputText value="#{projetoBean.projeto.dataInicio}">
18 <f:convertDateTime pattern="dd/MM/yyyy" type="date" />
19 </h:inputText>
```



5





Você está em

DevMedia

```
28 | <h:commandButton action="#{projetoBean.salvar}" value="Enviar" /  
29 | <input type="reset" value="Limpar" />  
30 | </h:form>
```

Para concluir a implementação da *view*, vamos criar a página que será exibida caso a operação de persistência seja realizada com sucesso, ou seja, o preenchimento do formulário não viole as restrições aplicadas à classe **Projeto**. O código da página *sucesso.xhtml* pode ser visto na **Listagem 6**.

Listagem 6. Código da página *sucesso.xhtml*.

```
1 | <h:body>  
2 |   <f:view>  
3 |     Cadastro realizado com sucesso  
4 |   </f:view>  
5 | </h:body>
```

Configurando o Hibernate e a aplicação

Nesse momento vamos voltar nossa atenção à configuração do Hibernate, onde devemos informar os detalhes para acesso ao banco de dados, assim como realizar o mapeamento da nossa classe de domínio, o que deve ser feito em um arquivo XML.



5





Você está em

DevMedia

Listagem 7. Conteúdo do arquivo hibernate.cfg.xml.

```
1  <hibernate-configuration>
2    <session-factory>
3      <property name="hibernate.dialect">org.hibernate.dialect.MySQLDial
4      <property name="hibernate.connection.driver_class">com.mysql.jdbc.
5      <property name="hibernate.connection.url">jdbc:mysql://localhost:3
6      <property name="hibernate.connection.username">root</property>
7      <property name="hibernate.connection.password">1234</property>
8      <property name="hibernate.show_sql">true</property>
9      <mapping class="com.jm.entidade.Projeto" />
10     </session-factory>
11  </hibernate-configuration>
```

As propriedades que configuramos são explicadas a seguir:

- **dialect (linha 5):** define o dialeto/linguagem com o qual o Hibernate se comunicará com a base de dados;
- **connection.driver_class (linha 6):** configura a classe do driver JDBC;
- **connection.url (linha 7):** determina a URL de conexão com o banco de dados;
- **connection.username (linha 8):** local onde deve ser informado o nome do usuário para conexão com o banco;
- **connection.password (linha 9):** local onde deve ser informada a senha;
- **show_sql (linha 10):** opção que possibilita visualizarmos o script SQL gerado pelo Hibernate;



5





Você está em

DevMedia

WEB-INF, selecione *Novo > Documento XML* e defina seu nome. A **Listagem 8** mostra o conteúdo desse arquivo.

Note que dentro da tag **<welcome-file>**, na **linha 2**, é definida a página que será tida como inicial pela aplicação. Já entre as **linhas 8 e 12** é configurado o *servlet* do JSF, e entre as **linhas 13 e 16** é especificado o padrão de URL através do qual o Servlet, dado em **<servlet-name>**, pode ser acessado.

Já o parâmetro **javax.faces.VALIDATE_EMPTY_FIELDS**, vide linha 5, quando configurado como **true**, força o JSF a acionar a validação de campos vazios, pois por padrão o JSF trata os campos vazios como nulos.

Listagem 8. Conteúdo do arquivo web.xml.

```
1 <welcome-file-list>
2   <welcome-file>project_form.xhtml</welcome-file>
3 </welcome-file-list>
4 <context-param>
5   <param-name>javax.faces.VALIDATE_EMPTY_FIELDS</param-name>
6   <param-value>true</param-value>
7 </context-param>
8 <servlet>
9   <servlet-name>Faces Servlet</servlet-name>
10  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
11  <load-on-startup>1</load-on-startup>
12 </servlet>
13 <servlet-mapping>
14   <servlet-name>Faces Servlet</servlet-name>
15   <url-pattern>*.xhtml</url-pattern>
```



5





Você está em

DevMedia

implementar os métodos de controle das conexões e transações com o banco. Seu código fonte pode ser visto na **Listagem 9**.

Observe que nessa classe temos apenas o atributo **sessionFactory** e o método estático **getSessionFactory()**, que cria uma **SessionFactory** para o Hibernate de acordo com o arquivo de configurações. A partir disso, é possível instanciar objetos do tipo **org.hibernate.Session** que, por sua vez, serão utilizados para realizar as tarefas de persistência do framework, como apresentado a seguir.

Listagem 9. Código da classe HibernateUtil.

```
1 package util;
2
3 //imports omitidos
4
5 public class HibernateUtil {
6
7     private static SessionFactory sessionFactory;
8
9     public static SessionFactory getSessionFactory() {
10         if (sessionFactory == null) {
11             Configuration configuration = new Configuration().configure();
12             ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder
13                 .applySettings(configuration.getProperties()).build
14             sessionFactory = configuration.buildSessionFactory(serviceRegistry);
15             SchemaUpdate se = new SchemaUpdate(configuration);
16             se.execute(true, true);
17         }
18
19         return sessionFactory;
```



5





Você está em

DevMedia

Com a classe **Projeto** mapeada, os arquivos de configuração definidos e a classe auxiliar implementada, vamos codificar as operações de persistência. Para isso, crie a interface **ProjetoDao** em um pacote de nome **br.com.jm.projeto.dao**.

Para criar o pacote, clique com o botão direito do mouse sobre o projeto *hibernate-validator*, escolha *New > Package* e informe seu nome. Em seguida, clique sobre ele, novamente com o botão direito, acesse *New > Interface* e dê o nome de **ProjetoDao**. O código fonte deve ficar semelhante ao apresentado na **Listagem 10**. Nele, podemos verificar todas as operações que serão realizadas sobre o banco de dados.

Listagem 10. Código da interface ProjetoDao.

```
1 package br.com.jm.projeto.dao;
2
3 //imports omitidos
4
5 public interface ProjetoDao {
6     public void save(Projeto projeto);
7     public Projeto getProjeto(long id);
8     public List<Projeto> list();
9     public void remove(Projeto projeto);
10    public void update(Projeto projeto);
11 }
```

Feito isso, precisamos criar a classe para implementar essa interface. Assim, com



5





Você está em

DevMedia

```
1 package br.com.jm.projeto.dao;
2
3 //imports omitidos
4
5 public class ProjetoDaoImp implements ProjetoDao {
6
7     public void save(Projeto projeto) {
8         Session session = HibernateUtil.getSessionFactory().openSession();
9         Transaction t = session.beginTransaction();
10         session.save(projeto);
11         t.commit();
12     }
13
14     public Projeto getProjeto(long id) {
15         Session session = HibernateUtil.getSessionFactory().openSession();
16         return (Projeto) session.load(Projeto.class, id);
17     }
18
19     public List<Projeto> list() {
20         Session session = HibernateUtil.getSessionFactory().openSession();
21         Transaction t = session.beginTransaction();
22         List lista = session.createQuery("from Projeto").list();
23         t.commit();
24         return lista;
25     }
26
27     public void remove(Projeto projeto) {
28         Session session = HibernateUtil.getSessionFactory().openSession();
29         Transaction t = session.beginTransaction();
30         session.delete(projeto);
31         t.commit();
32     }
33
34     public void update(Projeto projeto) {
35         Session session = HibernateUtil.getSessionFactory().openSession();
```



5





Você está em

DevMedia

Vejamos uma análise das linhas mais importantes dessa classe:

- **Linha 7:** O método **save()** recebe como parâmetro os dados do projeto e realiza a inserção deste no banco de dados;
- **Linha 8:** A variável do tipo **Session** recebe o resultado da chamada ao método **openSession()**, que é chamado a partir do retorno do método **getSessionFactory()**;
- **Linha 9:** A operação **beginTransaction()** inicia uma transação com o banco de dados;
- **Linha 10:** O método **save()** realiza a persistência do objeto;
- **Linha 11:** O método **commit()** finaliza a transação e a sessão é encerrada.

Os demais métodos dessa classe (**list()**, **remove()** e **update()**) são semelhantes ao **save()** e por isso analisaremos aqui apenas o que há de diferente no código de cada um. Vejamos:

- **Linha 22:** O método **createCriteria()** especifica uma query para recuperar todos os objetos do tipo **Projeto** presentes no banco de dados;
- **Linha 30:** O método **delete()** remove o objeto passado como parâmetro;
- **Linha 37:** O método **update()** realiza a atualização do objeto passado como parâmetro.

Testando a aplicação



5





Você está em

DevMedia

clique no botão *Enviar* sem preencher qualquer campo. Desta forma, podemos verificar se a validação de preenchimento obrigatório está funcionando.

A **Figura 8** mostra o resultado desse teste. Note que como nenhuma informação foi inserida nos campos do formulário, o sistema alerta o usuário com algumas mensagens de erro.

The screenshot shows a web browser window with the title 'Novo Projeto'. The address bar displays 'localhost:8080/hibernate-validator/prc'. The main content area has the heading 'Preencha o formulário abaixo' and a list of error messages in red text:

- Informe o nome do Projeto
- Informe o nome do Responsável pelo Projeto
- Informe o número do CPF
- CPF inválido
- Email inválido

Below the errors are several input fields:

- Nome do Projeto:
- Responsável:
- CPF do Responsável:
- Email do Responsável:
- Data de Início do Projeto:
- Data de Término do Projeto:
- Descrição:

At the bottom are two buttons: 'Enviar' and 'Limpar'.

Figura 8. Formulário de cadastro de projeto em branco.



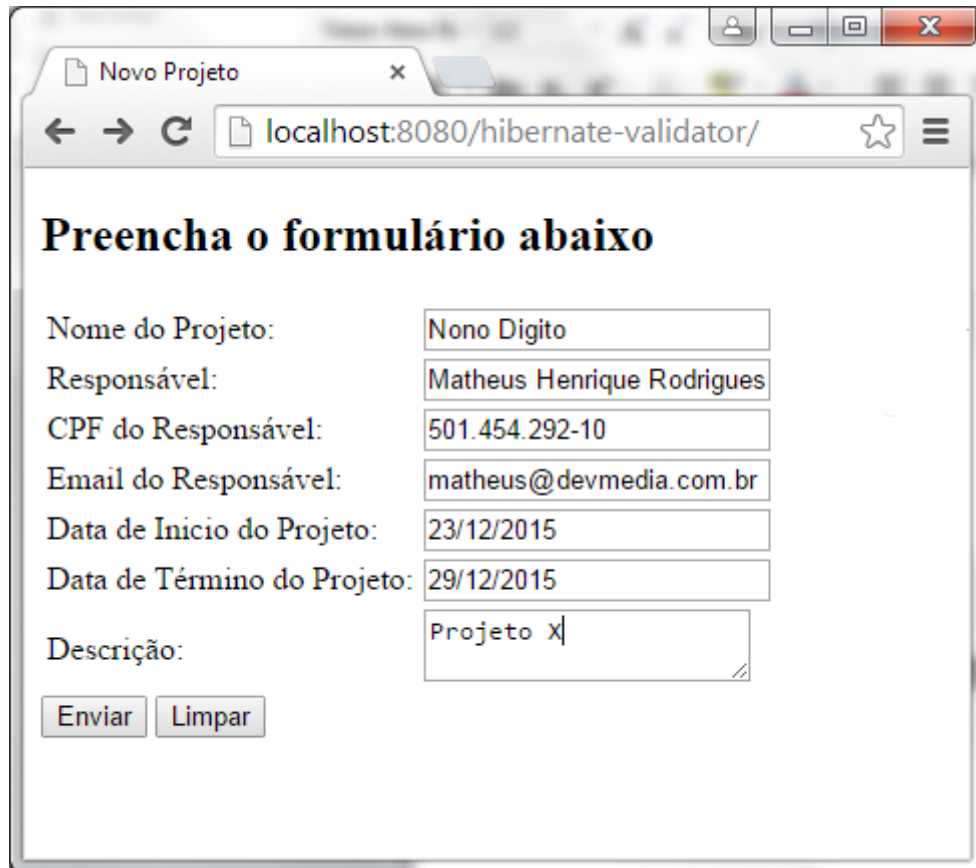
5





Você está em

DevMedia



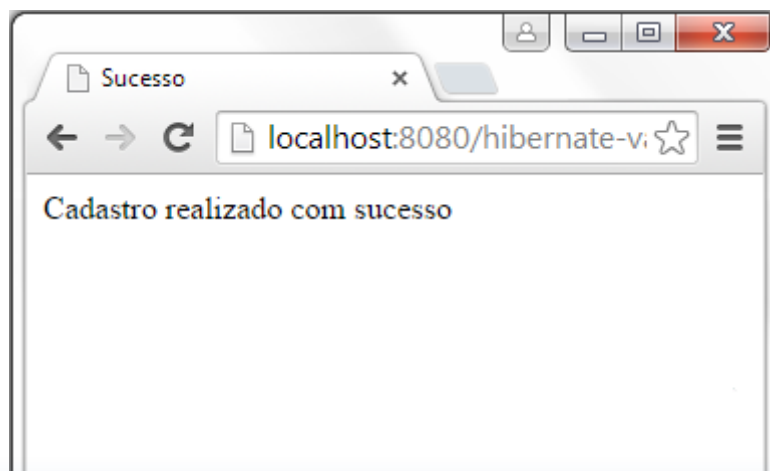
Novo Projeto

localhost:8080/hibernate-validator/

Preencha o formulário abaixo

Nome do Projeto:	Nono Digito
Responsável:	Matheus Henrique Rodrigues
CPF do Responsável:	501.454.292-10
Email do Responsável:	matheus@devmedia.com.br
Data de Inicio do Projeto:	23/12/2015
Data de Término do Projeto:	29/12/2015
Descrição:	Projeto x

Figura 9. Formulário de cadastro de projeto preenchido corretamente.



Sucesso

localhost:8080/hibernate-v...

Cadastro realizado com sucesso



5





Você está em

DevMedia

Hibernate Validator, é uma das variações existentes no modelo de domínio da aplicação, de forma centralizada, evitando assim a reescrita de código e tornando-o mais legível, o que facilita o trabalho do desenvolvedor durante tarefas de implementação de novos recursos e manutenção.

Por fim, vale ressaltar que diversos frameworks para desenvolvimento web já fazem integração com a Bean Validation, como é o caso do próprio Hibernate, utilizado em nosso exemplo, do JPA, Spring e JSF. Isso mostra a importância, o alcance e o nível de flexibilidade da API.

Links

- [Site do Hibernate](#)
- [Site do Hibernate Validator](#)
- [Site do MySQL](#)
- [Endereço para download do driver do MySQL](#)
- [Endereço para download do Eclipse](#)
- [Endereço para download do JDK](#)
- [Endereço para download do Maven](#)

Tecnologias:

Hibernate

Java

Maven

MySQL

SQL

Tomcat

XML



Marcar como concluído



Anotar



5





Você está em

DevMedia

Suporte ao aluno - Tire a sua dúvida.



Poste aqui a sua dúvida, nessa seção só você e o consultor podem ver os seus comentários.

Enviar dúvida

Planos de estudo

Fale conosco

Assinatura para empresas

Assine agora



Hospedagem web por Porta 80 Web Hosting



5

