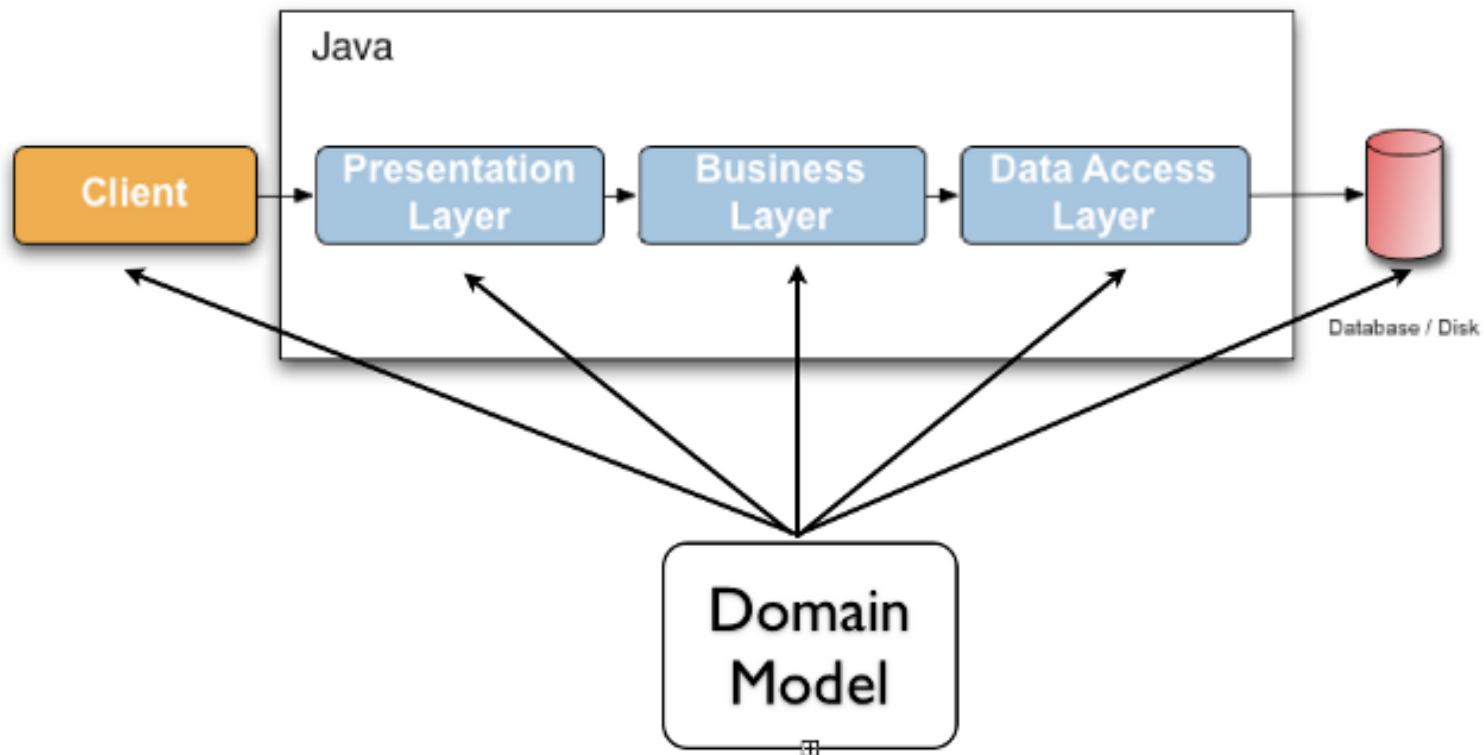




Bean Validation

JSR 380 - Bean Validation 2.0

O JSR 380 - Bean Validation 2.0 define um modelo de metadados e uma interface de programação (API) para a validação de entidades.




Configurando a Validação

```
@Entity
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer idCliente;
    @Size(min=3, max=150,
        message="O Nome deve ter no no mínimo 3 e no máximo 150 caracteres")
    private String nome;
    @Logradouro(max=150, message="O Endereço é inválido")
    private String endereco;
    @Pattern(regexp="(9[0-9]{4}|[1-9]{4})-[0-9]{4}",
        message="Nº de Telefone inválido")
    private String telefone;
    @Email
    private String email;
    private boolean desativado;
    @ManyToMany(fetch=FetchType.EAGER)
    @JoinTable(name="ServicosPrestados",
        joinColumns= {@JoinColumn(name="idCliente")},
        inverseJoinColumns={@JoinColumn(name="idServico")})
    private List<Servico> servicos;
}
```

Configurando a Validação

```
@Entity
public class Servico {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer idServico;
    @Size(min=3, max=150,
        message="O Nome deve ter no no mínimo 3 e no máximo 150 caracteres")
    private String nome;
    private boolean desativado;
    @Transient
    private boolean selecionado;
}
```



Construindo um Validador

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = LogradouroValidator.class)
@Documented
public @interface Logradouro {
    int max() default 0;

    String message() default "";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };
}
```

A implementação do Validador

```
public class LogradouroValidator implements ConstraintValidator<Logradouro, String> {
    private Logradouro annotation;

    @Override
    public void initialize(Logradouro annotation) {
        this.annotation = annotation;
    }

    @Override
    public boolean isValid(String valor, ConstraintValidatorContext ctx) {
        int max = annotation.max();

        if(valor == null || valor.length() > max) {
            return false;
        } else {
            String[] parte = valor.split(" ");
            if((parte[0].equalsIgnoreCase("rua") ||
                parte[0].equalsIgnoreCase("av.") ||
                parte[0].equalsIgnoreCase("estrada")) &&
                parte[1] != null) {
                return true;
            } else {
                return false;
            }
        }
    }
}
```

@DecimalMax(value=, inclusive=) - Verifica se o item anotado é menor ou igual ao valor especificado.

@DecimalMin(value=, inclusive=) - Verifica se o item anotado é maior ou igual ao valor especificado.

@Digits(integer=, fraction=) - Verifica se o item anotado tem a quantidade de dígitos até o especificado.

@Future - Verifica se o item anotado tem a data maior que a data especificada.

@Max(value=) - Verifica se o item anotado é menor ou igual ao valor especificado.

@Min(value=) - Verifica se o item anotado é maior ou igual ao valor especificado.

@NotNull - Verifica se o item anotado não é NULO.

@Past - Verifica se o item anotado tem a data anterior a data especificada.

@Pattern(regex=, flags=) - Verifica se o item anotado combina com a expressão regular especificada.

@Size(min=, max=) - Verifica se o item anotado tem o tamanho entre o especificado.

Tipos de Validação Adicionais

@CNPJ - Verifica se o item anotado representa um CNPJ.

@CPF - Verifica se o item anotado representa um CPF.

@TituloEleitoral - Verifica se o item anotado representa um Título Eleitoral.

@Email - Verifica se o item anotado representa um endereço de Email.


```
<div class="form-group">
  <label for="nome">Nome</label>
  <input class="form-control" type="text" th:field="*{nome}"
    required aria-required="true">
  <span th:if="${#fields.hasErrors('nome')}}" th:errors="*{nome}"
    th:errorclass="text-danger"></span>
</div>
<div class="form-group">
  <label for="endereco">Endereço</label>
  <input class="form-control" type="text" th:field="*{endereco}"
    required aria-required="true">
  <span th:if="${#fields.hasErrors('endereco')}}" th:errors="*{endereco}"
    th:errorclass="text-danger"></span>
</div>
<div class="form-group">
  <label for="telefone">Telefone</label>
  <input class="form-control" type="text" th:field="*{telefone}"
    required aria-required="true">
  <span th:if="${#fields.hasErrors('telefone')}}" th:errors="*{telefone}"
    th:errorclass="text-danger"></span>
</div>
<div class="form-group">
  <label for="email">E-Mail</label>
  <input class="form-control" type="text" th:field="*{email}"
    required aria-required="true">
  <span th:if="${#fields.hasErrors('email')}}" th:errors="*{email}"
    th:errorclass="text-danger"></span>
</div>
```

Identificando no Controller

@Valid para
validar o item

BindingResult para
identificar se houve erro

```
@RequestMapping("/cadastra")
public ModelAndView cadastra(@ModelAttribute("cliente") @Valid Cliente cliente, BindingResult result,
    RedirectAttributes redirectAttributes) {

    if(result.hasErrors()) {
        return new ModelAndView("editaCadastro", "cliente", cliente);
    } else {
        Integer id = cliente.getIdCliente();
        clienteDao.salvar(cliente);

        if(id == null) {
            redirectAttributes.addFlashAttribute("mensagem", "Cliente cadastrado");
            redirectAttributes.addFlashAttribute("tipo.mensagem", "alert-success");
            return new ModelAndView("redirect:/editaCadastro");
        } else {
            redirectAttributes.addFlashAttribute("mensagem", "Cliente atualizado");
            redirectAttributes.addFlashAttribute("tipo.mensagem", "alert-success");
            return new ModelAndView("redirect:/listaCadastro");
        }
    }
}
```

Se houve erro deve ser retornado para a página de origem a fim de apresentar as mensagens de erro

Novo Cliente

Nome

O Nome deve ter no no mínimo 3 e no máximo 150 caracteres

Endereço

O Endereço é inválido

Telefone

Nº de Telefone inválido

E-Mail

não é um endereço de e-mail

http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/



Hibernate Validator 6.0.12.Final - JSR 380 Reference Implementation: Reference Guide

Hardy Ferentschik · Gunnar Morling · Guillaume Smet – 2018-08-10

Preface

Validating data is a common task that occurs throughout all application layers, from the presentation to the persistence layer. Often the same validation logic is implemented in each layer which is time consuming and error-prone. To avoid duplication of these validations, developers often bundle validation logic directly into the domain model, cluttering domain classes with validation code which is really metadata about the class itself.