Tarefa 2 | Prática de TDD, Refactoring usando Caso de Ensino

UNISINOS

Integrantes

- Felipe Griep
- Giulia Cardoso
- Rosival de Souza
- Tiago Zardin

UNISINOS

Requisito escolhido

RF03 – Cadastro de Produtos

O sistema deve possibilitar o cadastro de insumos e produtos finais.

UNISINOS

História de Usuário

Como **analista de produção**, quero **cadastrar novos produtos e insumos**, para que **eles possam ser rastreados e utilizados nos processos de produção**.

Cenário 1: Cadastro de um produto válido

Dado que o usuário acessa a tela de cadastro de produtos

Quando ele preenche corretamente os campos obrigatórios (código, nome, fornecedor e categoria) e confirma

Então o produto deve ser salvo no sistema e exibido na lista de produtos

UNISINOS

Cenário 2: Cadastro com campos obrigatórios em branco

Dado que o usuário acessa a tela de cadastro de produtos

Quando ele tenta salvar sem preencher os campos obrigatórios

Então o sistema deve exibir uma mensagem de erro informando que os campos são obrigatórios

UNISINOS

Cenário 3: Cadastro de produto já existente

Dado que já existe um produto com o mesmo código

Quando o usuário tenta cadastrar novamente o produto

Então o sistema deve impedir o cadastro e exibir mensagem de duplicidade

UNISINOS

Cenário 4: Edição de produto já cadastrado

Dado que existe um produto já registrado

Quando o usuário altera informações do produto e confirma

Então o sistema deve atualizar os dados e exibir o produto atualizado na lista

UNISINOS

Cenário 5: Exclusão de produto cadastrado

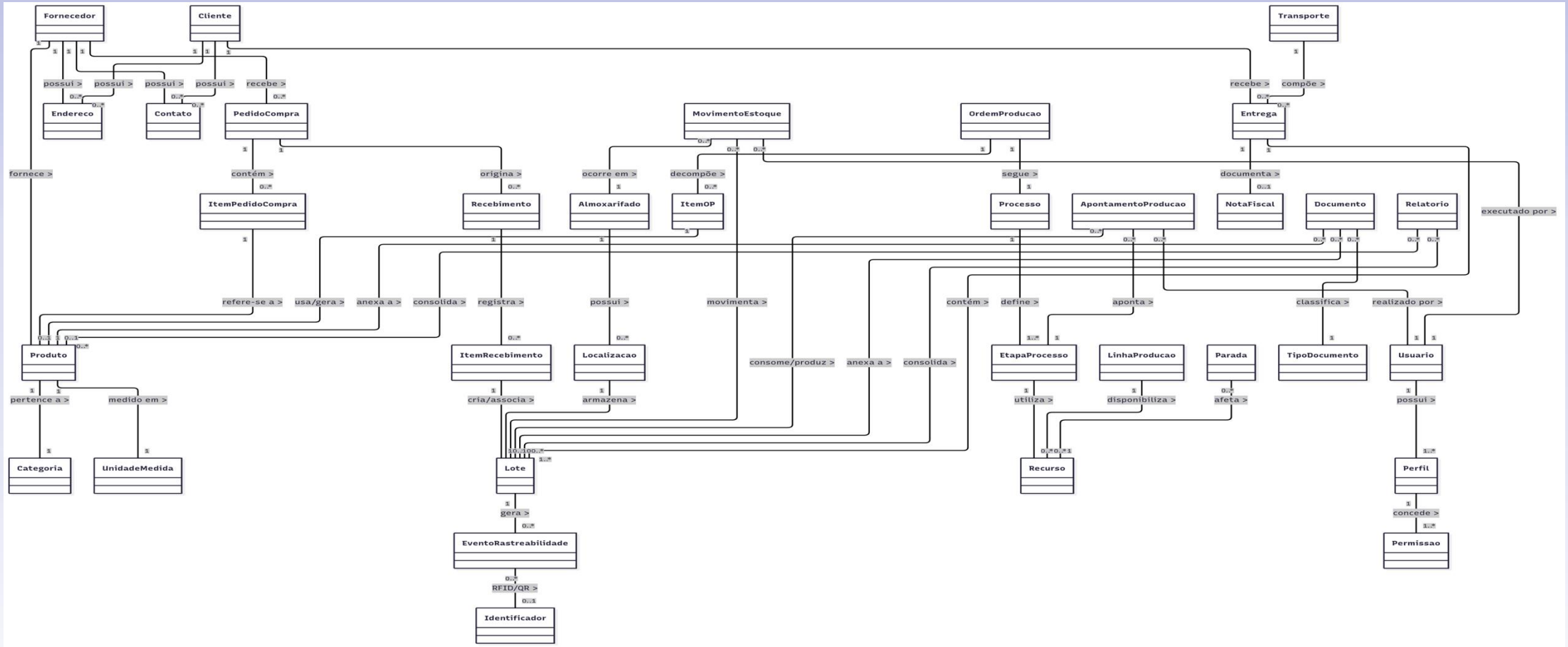
Dado que existe um produto registrado no sistema

Quando o usuário seleciona a opção de excluir e confirma a operação

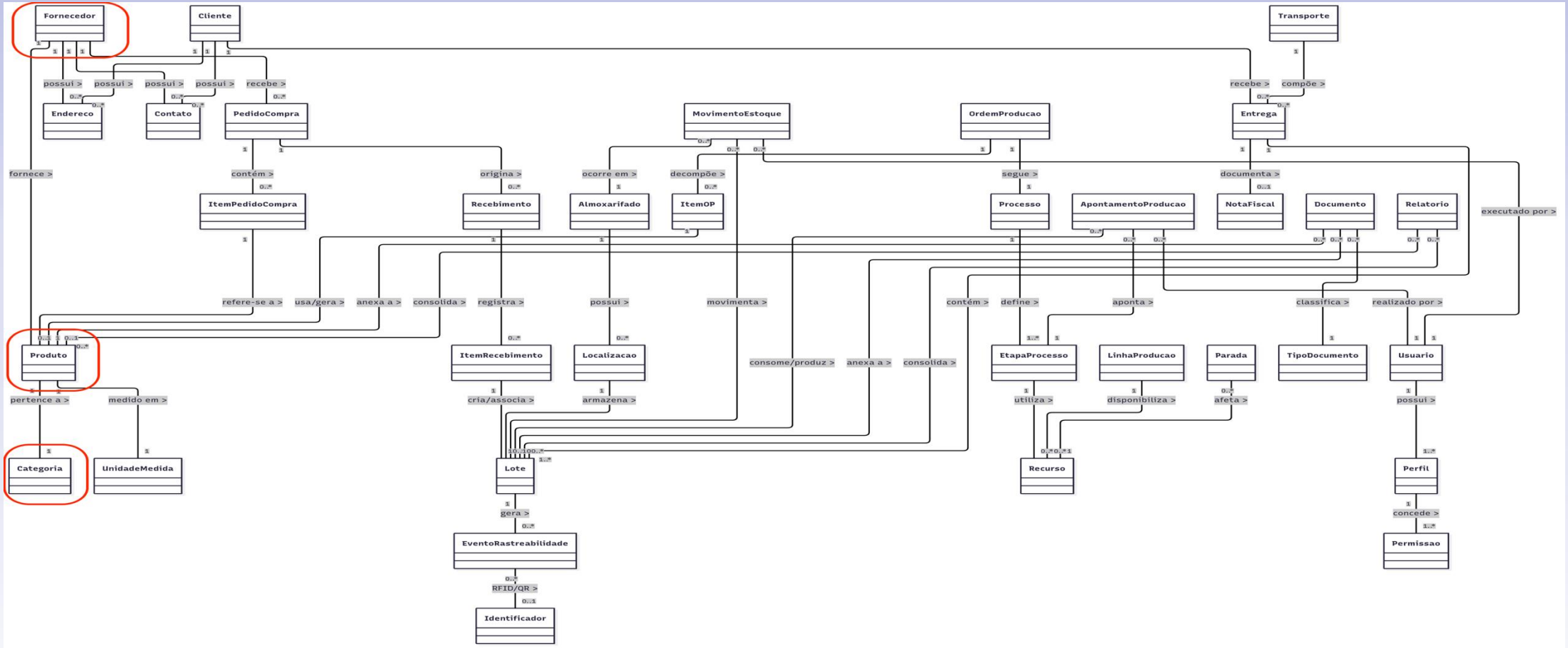
Então o sistema deve remover o produto da base de dados

UNISINOS

Diagrama de Classes

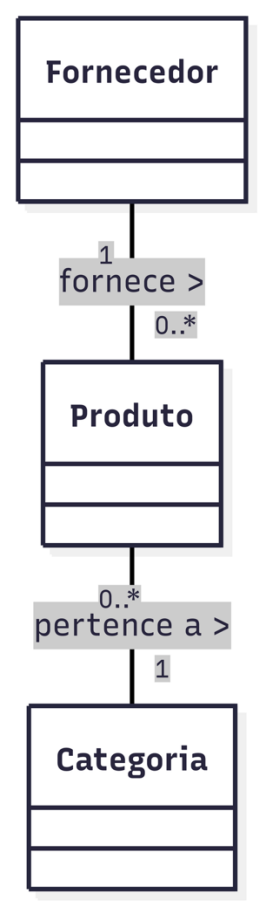


Seleção das Classes



UNISINOS

Classes Selecionadas



UNISINOS

Roteiro TDD - Cenário 1: Cadastro de um produto válido

- [x] 01. Criar a classe de Teste RepositorioProdutoTest;
- [x] 02. Criar o teste deveCadastrarUmProdutoValido, anotando com @Test;
- [x] 03. Criar uma instância do tipo Produto com os dados de código e nome;
- [x] 04. O teste deve falhar;
- [x] 05. Criar a classe Produto com os atributos código e nome;
- [x] 06. O teste deve passar;
- [...]
- [x] 30. O teste deve falhar;
- [x] 31. Criar a classe Categoria com os atributos código e nome;
- [x] 32. O teste deve passar;
- [x] 33. Passar a instância de Categoria para o construtor do Produto, como quarto parâmetro;
- [x] 34. O teste deve falhar;
- [x] 35. Alterar a classe Produto, adicionando o atributo do tipo Categoria, receber como quarto parâmetro no construtor e atribuir ao atributo da classe;
- [x] 36. O teste deve passar;

UNISINOS

Roteiro TDD - Cenário 2: Cadastro com campos obrigatórios em branco

- [x] 01. Criar o teste naoDeveCadastrarUmProdutoSemDadosObrigatorios, anotando com @Test;

- [x] 02. Criar uma instância do tipo Produto com o construtor sem parâmetros;

- [x] 03. O teste deve falhar;

- [x] 04. Alterar a classe Produto, criando um construtor sem parâmetros;

- [x] 05. O teste deve passar;

- [x] 06. Realizar a chamada de RepositorioProduto.cadastrarProduto() e passar a instância de Produto, verificando se o retorno é uma Exception do tipo IllegalArgumentException;

- [x] 07. O teste deve falhar;

- [x] 08. Alterar RepositorioProduto.cadastrarProduto(), incluindo validação de todos os atributos se possuem valor. Se algum for nulo ou vazio retorna um IllegalArgumentException com a mensagem "Dados obrigatórios não informados.". Deve ser importado o “org.apache.commons:commons-lang3:3.18.0” e utilizados os métodos StringUtils.isEmpty() e ObrjectUtils.isEmpty() para a validação;

- [x] 09. O teste deve passar;

- [x] 10. Alterar o teste naoDeveCadastrarUmProdutoSemDadosObrigatorios para validar se a mensagem da Exception é "Dados obrigatórios não informados.";

- [x] 11. O teste deve passar;

- [x] 12. Alterar o teste naoDeveCadastrarUmProdutoSemDadosObrigatorios para validar se o retorno RepositorioProduto.listarProdutos() é 0;

- [x] 13. O teste deve passar;

UNISINOS

Roteiro TDD - Cenário 3: Cadastro de produto já existente

- [x] 01. Criar o teste naoDeveCadastrarUmProdutoDuplicado, anotando com @Test;

- [x] 02. Criar uma instância do tipo Produto com os dados de código, nome, fornecedor e categoria;

- [x] 03. O teste deve passar;

- [x] 04. Realizar a chamada de RepositorioProduto.cadastrarProduto() e passar a instância de Produto, verificando se o retorno é true;

- [x] 05. O teste deve passar;

- [x] 06. Criar uma nova instância do tipo Produto com os dados de código, nome, fornecedor e categoria, mas com o mesmo código;

- [x] 07. O teste deve passar;

- [x] 08. Realizar a chamada de RepositorioProduto.cadastrarProduto() e passar a nova instância de Produto, verificando se o retorno é uma Exception do tipo IllegalArgumentException;

- [x] 09. O teste deve falhar;

- [x] 10. Alterar RepositorioProduto.cadastrarProduto(), incluindo, após a validação dos atributos, a validação da existência do código do produto na lista de produtos e se existir retorna um IllegalArgumentException com a mensagem “Produto XXX já cadastrado.”, onde XXX é o código do produto;

- [x] 11. O teste deve passar;

UNISINOS

Roteiro TDD - Cenário 4: Edição de produto já cadastrado

- [x] 01. Criar o teste deveEditarProdutoCadastrado, anotando com @Test;
- [x] 02. Criar uma instância do tipo Produto com os dados de código, nome, fornecedor e categoria;
- [x] 03. O teste deve passar;
- [x] 04. Realizar a chamada de RepositorioProduto.cadastrarProduto() e passar a instância de Produto, verificando se o retorno é true;
- [x] 05. O teste deve passar;
- [x] 06. Criar uma nova instância do tipo Produto com os dados de código, nome, fornecedor e categoria, mas com o mesmo código;
- [x] 07. O teste deve passar;
- [x] 08. Realizar a chamada de RepositorioProduto.editarProduto() e passar a instância de Produto, verificando se o retorno é true;
- [x] 09. O teste deve falhar;
- [x] 10. Criar o método editarProduto(), como static, retornando o tipo boolean com o valor true;
- [x] 11. O teste deve passar;
- [x] 12. Alterar o teste deveEditarProdutoCadastrado para validar se o último produto de RepositorioProduto.listarProdutos() é igual a nova instância de Produto;
- [x] 13. O teste deve falhar;
- [x] 14. Alterar o método RepositorioProduto.editarProduto(), incluindo a lógica para atualizar o produto;
- [x] 15. O teste deve passar;

UNISINOS

Roteiro TDD - Cenário 5: Exclusão de produto cadastrado

- [x] 01. Criar o teste deveExcluirProdutoCadastrado, anotando com @Test;
- [x] 02. Criar uma instância do tipo Produto com os dados de código, nome, fornecedor e categoria;
- [x] 03. O teste deve passar;
- [x] 04. Realizar a chamada de RepositorioProduto.cadastrarProduto() e passar a instância de Produto, verificando se o retorno é true;
- [x] 05. O teste deve passar;
- [x] 06. Realizar a chamada de RepositorioProduto.excluirProduto() e passar o código do produto a ser excluído, verificando se o retorno é true;
- [x] 07. O teste deve falhar;
- [x] 08. Criar o método excluirProduto(), como static, retornando o tipo boolean com o valor true;
- [x] 09. O teste deve passar;
- [x] 10. Alterar o teste deveExcluirProdutoCadastrado para validar se o último produto de RepositorioProduto.listarProdutos() é não é igual a instância de Produto;
- [x] 11. O teste deve falhar;
- [x] 12. Alterar o método RepositorioProduto.excluirProduto(), incluindo a lógica para excluir o produto;
- [x] 13. O teste deve passar;

UNISINOS

Roteiro Refactoring

- [x] 01. Substituir a dependência “org.apache.commons:commons-lang3:3.18.0” pela dependência “jakarta.validation:jakarta.validation-api:3.1.1” e incluir as dependências “org.hibernate.validator:hibernate-validator:9.0.1.Final” e “org.glassfish.expressly:expressly:6.0.0”

- [x] 02. Incluir a dependência de testes “org.hamcrest:hamcrest:2.2”

- [x] 03. Remover o bloco de validações de vazio do método RepositorioProduto.cadastrarProduto();

- [x] 04. Alterar a classe Produto, incluindo “@NotBlank(message = "O atributo xxx e obrigatorio")” para os atributos tipo String e “@NotNull(message = "O atributo xxx e obrigatorio”)” para os atributos do tipo objeto. “xxx” deve ser usado o nome do atributo;

- [x] 05. Alterar a classe RepositórioProduto e criar uma variável do tipo “jakarta.validation.ValidatorFactory” e uma variável do tipo “jakarta.validation.Validator”;

[...]

- [x] 38. Os testes devem passar.

- [x] 39. No método static criarProduto() criar um parâmetro nome do tipo String, esse parâmetro deve ser atribuído ao atributo nome do Produto;

- [x] 40. Todos os testes irão falhar;

- [x] 41. Em todos as chamadas do método static criarProduto(), deve ser passado o parâmetro com o nome originalmente definido;

- [x] 42. Os testes devem passar;

- [x] 43. No teste naoDeveCadastrarUmProdutoDuplicado, substituir a criação na variável produto2 pelo método static criarProduto() passando o nome original como parâmetro;

- [x] 44. Todos os testes deve passar;

- [x] 45. Criar uma classe chamada ProdutoFactory, em nível de testes, e mover os métodos static para essa classe. Via funcionalidade da IDE;

- [x] 46. Todos os testes devem passar;

Tecnologias

- OpenJDK 25 – Amazon Corretto
- Gradle 9.1
- Git

Depêndencias:

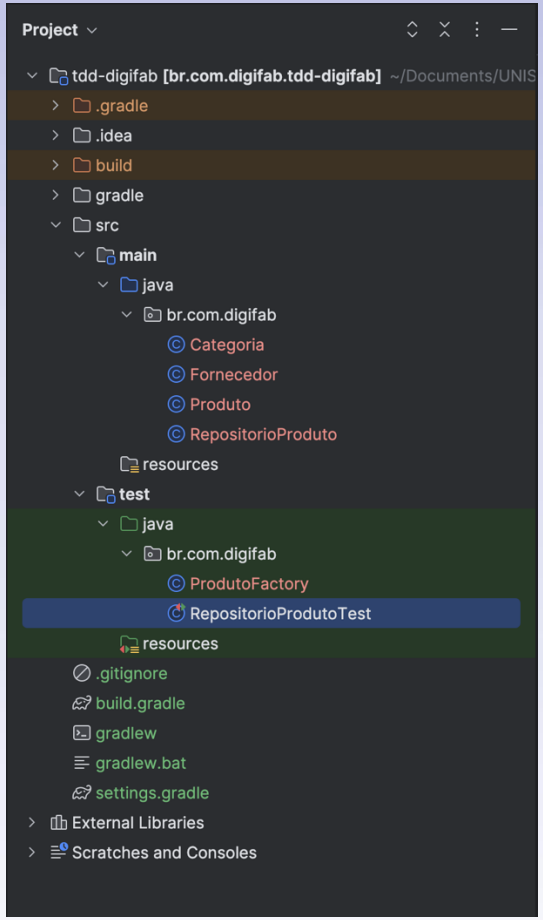
- org.junit:junit-bom:5.10.0
- org.junit.jupiter:junit-jupiter
- org.junit.platform:junit-platform-launcher:1.10.0

Adicionadas durante o Refactoring:

- org.hamcrest:hamcrest:2.2
- jakarta.validation:jakarta.validation-api:3.1.1
- org.hibernate.validator:hibernate-validator:9.0.1.Final
- org.glassfish.expressly:expressly:6.0.0

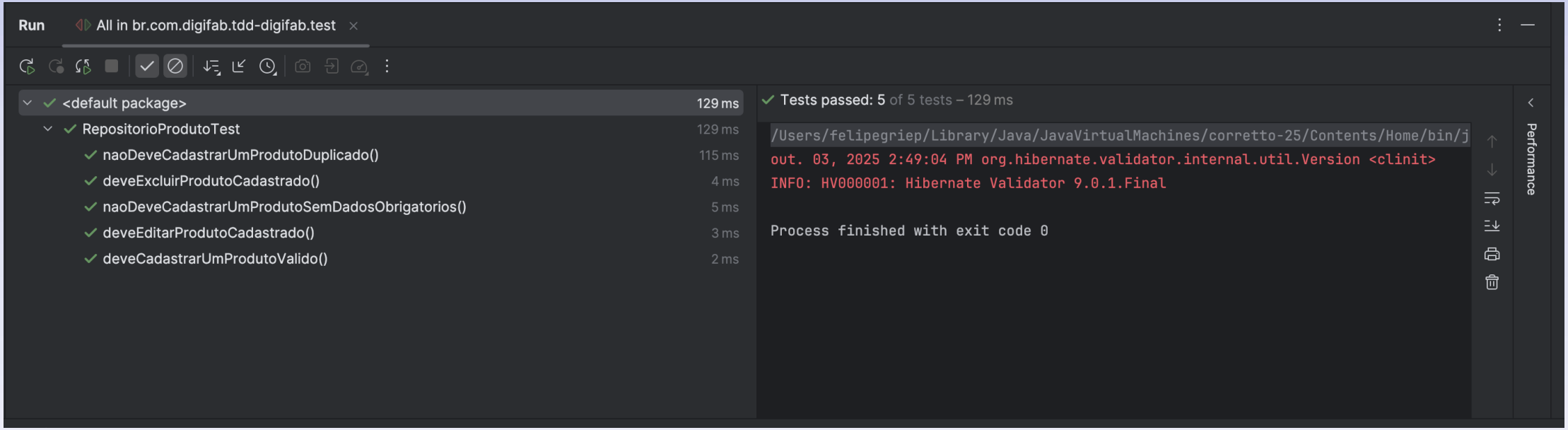
UNISINOS

Estrutura do Projeto



UNISINOS

Resultados dos Testes



UNISINOS

Conclusão

- O trabalho demonstrou como aplicar TDD e testes de aceitação para garantir qualidade desde o início do desenvolvimento;
- O diagrama conceitual permitiu visualizar a relação entre classes e apoiar a implementação;
- O maior desafio foi a escrita do roteiro;

UNISINOS

Obrigado!!!

UNISINOS