Estudo Dirigido 4 - Testes e Depuração

Lucas Apolonio de Amorim

15 de dezembro de 2024

1 Descrição do Programa

O objeto de estudo desse Estudo Dirigido é um programa de gerenciamento de estoque, que simula um banco de dados em menor escala e possui as seguintes classes e membros/funções:

- **Produto**

  id: ID do Produto.

  nome: Nome do Produto.

  preco: Preço do Produto.

  Produto(): Construtor do Objeto.

  getId(): Retorna o ID do Produto.

  getName(): Retorna o nome do Produto.

  getPrice(): Retorna o preço do Produto.

- **Catalogo**

  lista_produtos: Lista dos Produtos presentes no catálogo.

  adicionarProduto(): Adiciona um produto à lista de produtos no catálogo

  listarProdutos(): Lista todos os produtos presentes no catálogo ou avisa que o catálogo está vazio.

  buscarProdutoPorId(): Busca um produto que tenha o ID especificado. Retorna um Produto inválido caso não encontre nenhum.

- **Pedido**

  lista_produtos: Lista dos Produtos pedidos.

  nome_cliente: Nome do Cliente que realizou o Pedido.

  adicionarProduto(): Adiciona produtos ao Pedido

  calcularTotal(): Calcula o Valor total do Pedido

2 Decisões Técnicas

1. Os membros da classe produto são privados e só podem ser acessados pelo seu respectivo *getter*.
2. O Construtor da classe *Produto* joga exceções para:
   - ID negativo.
   - Nome vazio.
   - Preço Negativo.
3. O Catálogo só pode Conter Produtos com ID's diferentes.
4. O Catálogo começa vazio e tem seus produtos inseridos 1 por 1.
5. A função *Catalogo::adicionarProduto()* retorna *true* se a inserção pôde ser realizada e *false* caso a inserção não tenha sido realizada (Devido à outro produto com o mesmo ID presente no Catálogo).
6. Caso o Catálogo esteja vazio, a função *listarProdutos()* notifica o usuário de que ele está vazio.
7. A função *buscarProdutoPorId()* retorna um valor do tipo *std::optional<Produto>*. Em que pode ser retornado um produto (cujo valor é obtido aplicando o método *optional::value()*) ou vazio. O retorno é distinguido pelo método embutido *optional::has_value()*.
8. O construtor da classe *Pedido* aceita o campo *nome_cliente* vazio, seja não passando esse campo para o construtor ou passando um string vazia.
9. A função *Pedido::adicionarProduto()* aceita produtos com ID repetido já que não há um contador da quantidade de um produto no pedido.
10. A função *calcularTotal* joga uma exceção em caso de overflow do valor total.

OBS$_1$: Como nenhuma das classes depende de algum recurso externo que não será testado não houve a necessidade de mocking.

OBS$_2$: Pelo porte do projeto, foi utilizada a saída padrão (*std::cout*) para o logging.