Instituto Infnet – MIT em Engenharia de software com .NET

Projeto da Disciplina Tecnologia .NET [25E2_2]

Aluno: Leandro Antunes Felipe Machado

StockTrack – Sistema de controle de estoque

O projeto propõe o desenvolvimento de um Sistema de Controle de Estoque para gerenciar produtos, categorias e movimentações de entrada e saída em um pequeno depósito. O objetivo é garantir o controle eficiente do inventário, facilitando o cadastro de produtos e categorias, o registro de movimentações (entrada e saída) e a consulta do saldo atual de cada item.

Repositório no GitHub: https://github.com/leoantunesfm/StockTrack

Escopo e Problema a Ser Resolvido

Um pequeno depósito armazena e controla as entradas e saídas de itens dos seguimentos de eletrônicos, alimentos e vestuário. Hoje não é possível saber com exatidão a quantidade de cada item que existe no estoque, e não é possível rastrear as movimentações que foram realizadas.

Nesse contexto, o sistema precisa incluir novos produtos, registrar suas movimentações de entrada e saída, atualizando seu saldo em estoque e registrando um histórico de cada movimentação.

No cadastro do produto, é preciso vincular uma categoria como maneira de classificação dos itens. Além disso, cada seguimento tem uma característica específica que precisa ser registrada. Para eletrônicos temos a voltagem, para alimentos a data de validade e para vestuário o tamanho.

Modelagem do Domínio

Ubiquitous Language:

- **Produto:** Item específico com características individuais e controlado no estoque.
- **Tipo:** Define o seguimento do produto limitado no contexto a Eletrônico, Alimento e Vestuário.
- **Categoria:** Classificação do produto dentro do seu **Tipo**. (Exemplo: Eletrônico – Smartphones, Alimento – Cereais, Vestuário – Masculino.
- **Movimentação:** Um registro de entrada ou saída de quantidade de itens do estoque, com data e hora.
- **Estoque:** Quantidade atual de itens de um produto armazenadas no depósito.

- **Voltagem/Data de Validade/Tamanho:** Características específicas de cada **Tipo** de produto.
- **Código curto:** Identificador único e amigável o **Produto**.

Entities, Value Objects e Repositories

Entities

- **Produto** (abstrata):
  - Propriedades: Id, CodigoCurto, Nome, Categoria, Estoque
  - Subtipos: Eletronico, Alimento, Vestuario
- **Categoria**:
  - Propriedades: Id, Nome
- **Movimentacao**:
  - Propriedades: Id, ProdutoId, Produto, Quantidade, Tipo (Entrada/Saída), Data

Value Objects

·**NomeProduto**: Valor do nome do produto (com validação de tamanho)

·**Quantidade**: Valor inteiro representando a quantidade

·**TipoMovimentacao**: Enum (Entrada, Saída)

Repositories

·**IProdutoRepository**: Interface para persistência e consulta de produtos

·**ICategoriaRepository**: Interface para persistência e consulta de categorias

·**IMovimentacaoRepository**: Interface para persistência e consulta de movimentações

Aggregates, Bounded Contexts e Domain Services

Aggregates

·**Produto** é a raiz de agregação para produtos e suas características específicas.

·**Categoria** é um aggregate independente.

·**Movimentacao** é um aggregate relacionado a Produto.

Bounded Contexts

- **Contexto de Produtos:** Responsável pelo ciclo de vida dos produtos e suas categorias.
- **Contexto de Movimentações:** Responsável pelos registros de entrada e saída, histórico e validações de movimentações.

Cada contexto possui seu próprio modelo e responsabilidades, mas interagem entre si através de interfaces bem definidas.

Domain Services

·**MovimentacaoService**:
Serviço de domínio responsável por aplicar regras de negócio relacionadas à movimentação de estoque (ex: validação de saldo, registro de movimentação).

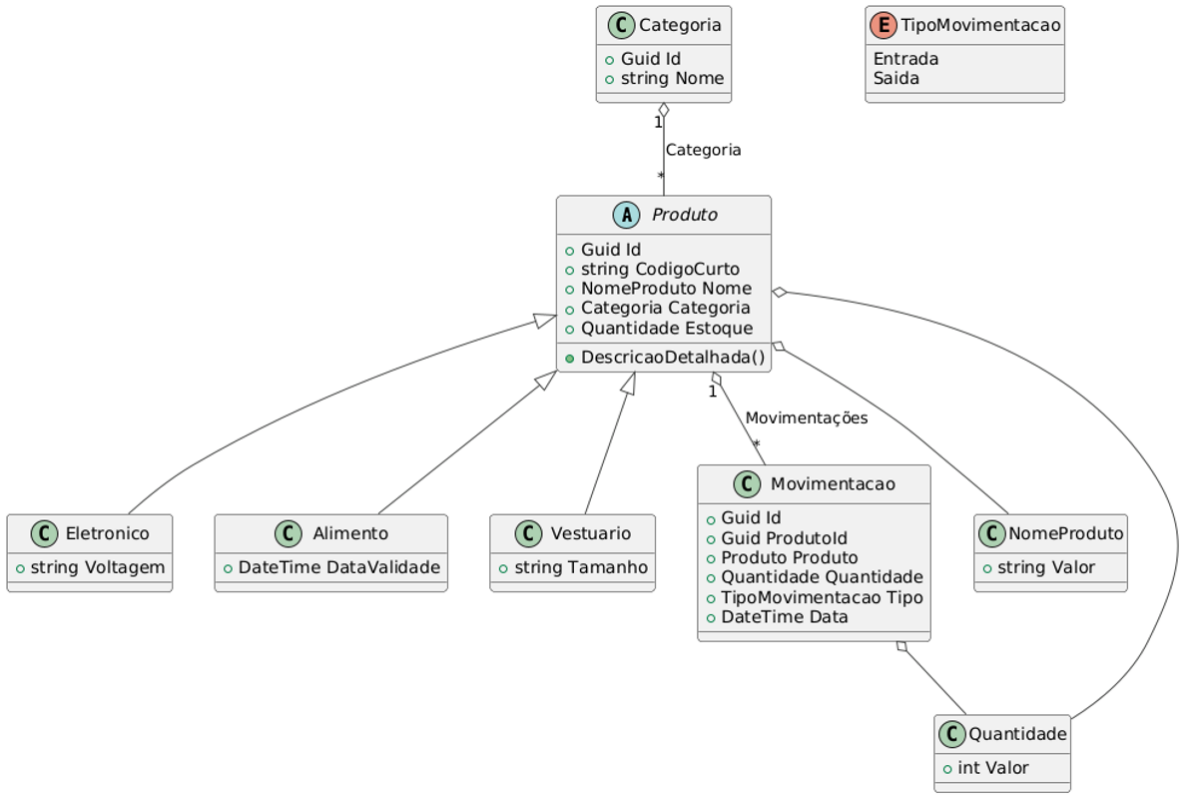
·**BuscaAvancadaProdutoService**:
Serviço de domínio para buscas complexas de produtos, utilizando critérios de nome e categoria.

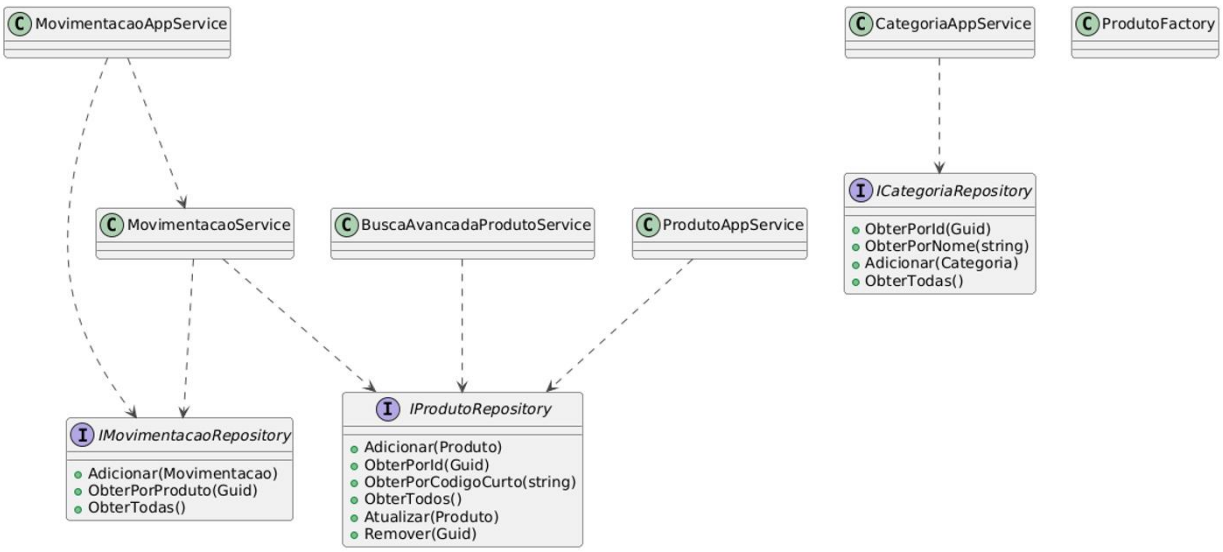
Domain Services vs Factories

Domain Services: Encapsulam regras de negócio que não pertencem a uma única entidade, como movimentação de estoque e buscas avançadas.

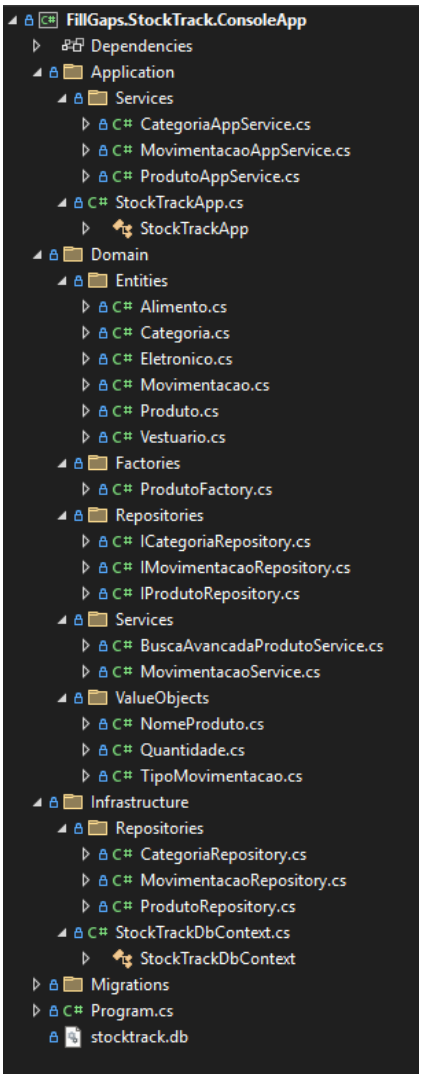
Factories: Responsáveis por criar instâncias de entidades complexas, como a ProdutoFactory, que instancia corretamente os diferentes tipos de produto com suas características específicas.

Diagrama da Modelagem do Domínio

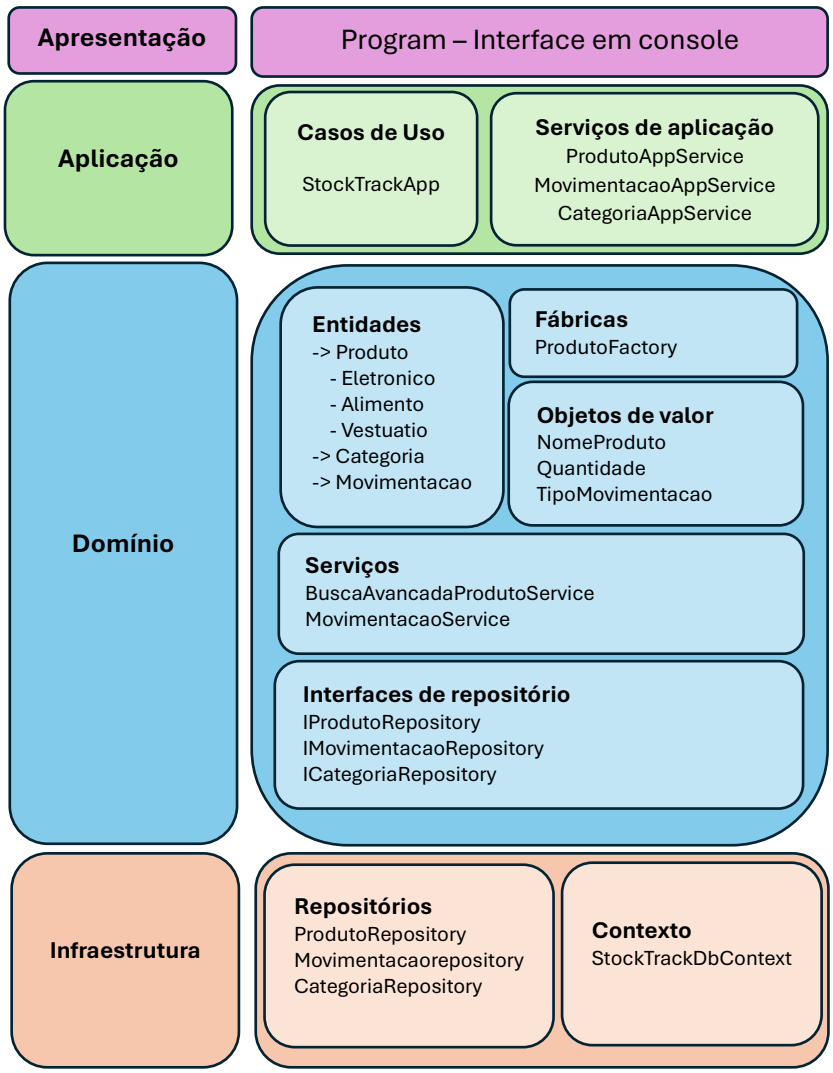




Estrutura do projeto



Arquitetura em camadas

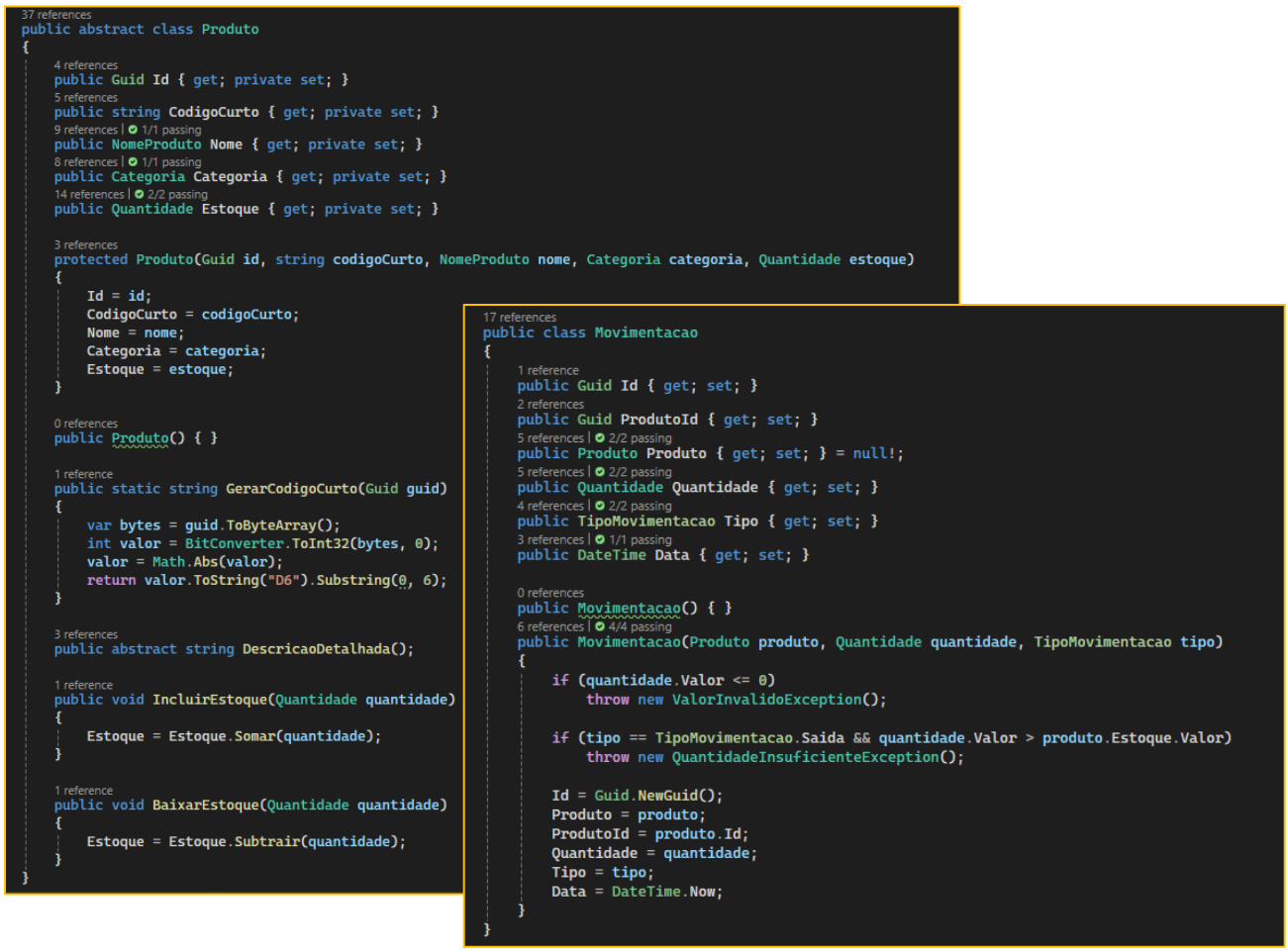


Aplicação de Padrões de Projeto – SOLID e GRASP

Princípios do padrão SOLID:

Single Responsibility

As classes de entidade Produto e Movimentacao são específicas e lidam apenas com regras de negócio inerentes ao seu domínio e alterações de estado:



```
37 references
public abstract class Produto
{
    4 references
    public Guid Id { get; private set; }
    5 references
    public string CodigoCurto { get; private set; }
    9 references | 1/1 passing
    public NomeProduto Nome { get; private set; }
    8 references | 1/1 passing
    public Categoria Categoria { get; private set; }
    14 references | 2/2 passing
    public Quantidade Estoque { get; private set; }

    3 references
    protected Produto(Guid id, string codigoCurto, NomeProduto nome, Categoria categoria, Quantidade estoque)
    {
        Id = id;
        CodigoCurto = codigoCurto;
        Nome = nome;
        Categoria = categoria;
        Estoque = estoque;
    }

    0 references
    public Produto() { }

    1 reference
    public static string GerarCodigoCurto(Guid guid)
    {
        var bytes = guid.ToByteArray();
        int valor = BitConverter.ToInt32(bytes, 0);
        valor = Math.Abs(valor);
        return valor.ToString("D6").Substring(0, 6);
    }

    3 references
    public abstract string DescricaoDetalhada();

    1 reference
    public void IncluirEstoque(Quantidade quantidade)
    {
        Estoque = Estoque.Somar(quantidade);
    }

    1 reference
    public void BaixarEstoque(Quantidade quantidade)
    {
        Estoque = Estoque.Subtrair(quantidade);
    }
}
```

```
17 references
public class Movimentacao
{
    1 reference
    public Guid Id { get; set; }
    2 references
    public Guid ProdutoId { get; set; }
    5 references | 2/2 passing
    public Produto Produto { get; set; } = null!;
    5 references | 2/2 passing
    public Quantidade Quantidade { get; set; }
    4 references | 2/2 passing
    public TipoMovimentacao Tipo { get; set; }
    3 references | 1/1 passing
    public DateTime Data { get; set; }

    0 references
    public Movimentacao() { }
    6 references | 4/4 passing
    public Movimentacao(Produto produto, Quantidade quantidade, TipoMovimentacao tipo)
    {
        if (quantidade.Valor <= 0)
            throw new ValorInvalidoException();

        if (tipo == TipoMovimentacao.Saida && quantidade.Valor > produto.Estoque.Valor)
            throw new QuantidadeInsuficienteException();

        Id = Guid.NewGuid();
        Produto = produto;
        ProdutoId = produto.Id;
        Quantidade = quantidade;
        Tipo = tipo;
        Data = DateTime.Now;
    }
}
```

Outros exemplos de classe que aplicam esse princípio:

ProdutoAppService: Responsável apenas por orquestrar operações de aplicação relacionadas a produtos.

MovimentacaoService: Responsável apenas pelas regras de negócio de movimentação de estoque.

Open/Closed

Produto é uma classe abstrata. Para adicionar um novo tipo de produto (ex: "Brinquedo"), basta criar uma nova subclasse (ex: Brinquedo : Produto) e adicionar no **ProdutoFactory**, sem alterar as subclasses existentes ou mesmo a classe Produto.

Os repositórios implementam interfaces, permitindo novas implementações ou alterações de persistência sem alterar o código que os consome.

Liskov Substitution

A Classe **MovimentacaoService** que é responsável por registrar as entradas e saídas em estoque, recebe nos métodos que executam a ação apenas a classe base, e executa a operação sem erros para qualquer tipo de produto:

```
4 references
public class MovimentacaoService
{
    private readonly IProdutoRepository _produtoRepository;
    private readonly IMovimentacaoRepository _movimentacaoRepository;

    1 reference
    public MovimentacaoService(IProdutoRepository produtoRepository, IMovimentacaoRepository movimentacaoRepository)
    {
        _produtoRepository = produtoRepository;
        _movimentacaoRepository = movimentacaoRepository;
    }

    1 reference
    public void RegistrarEntrada(Produto produto, Quantidade quantidade)
    {
        produto.IncluirEstoque(quantidade);

        var movimentacao = new Movimentacao(produto, quantidade, TipoMovimentacao.Entrada);
        _movimentacaoRepository.Adicionar(movimentacao);

        _produtoRepository.Atualizar(produto);
    }

    1 reference
    public void RegistrarSaida(Produto produto, Quantidade quantidade)
    {
        produto.BaixarEstoque(quantidade);

        var movimentacao = new Movimentacao(produto, quantidade, TipoMovimentacao.Saida);
        _movimentacaoRepository.Adicionar(movimentacao);

        _produtoRepository.Atualizar(produto);
    }
}
```

Interface Segregation

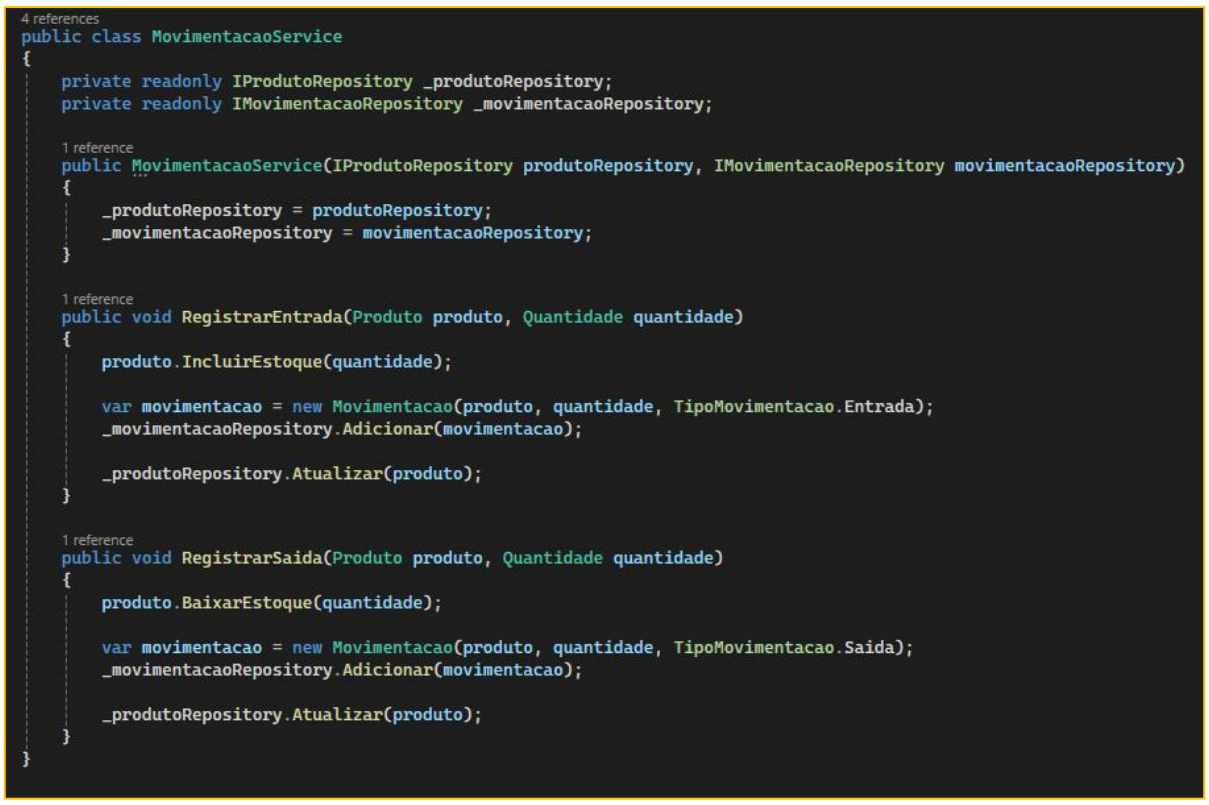
As interfaces de repositório contêm apenas métodos relevantes para a respectiva entidade. Por exemplo, IMovimentacaoRepository não obriga a implementação de métodos de produto ou categoria.

```
7 references
public interface IProdutoRepository
{
    2 references
    void Adicionar(Produto produto);
    2 references
    Produto? ObterPorId(Guid id);
    2 references
    Produto? ObterPorCodigoCurto(string codigoCurto);
    3 references
    IEnumerable<Produto> ObterTodos();
    3 references
    void Atualizar(Produto produto);
    1 reference
    void Remover(Guid id);
}
```

```
5 references
public interface IMovimentacaoRepository
{
    3 references
    void Adicionar(Movimentacao movimentacao);
    2 references
    IEnumerable<Movimentacao> ObterPorProduto(Guid produtoId);
    1 reference
    IEnumerable<Movimentacao> ObterTodas();
}
```

Dependency Inversion

O serviço de domínio MovimentacaoService depende das interfaces IProdutoRepository e IMovimentacaoRepository, e não de implementações concretas. O serviço de domínio MovimentacaoService depende das interfaces IProdutoRepository e IMovimentacaoRepository, e não de implementações concretas.



```
4 references
public class MovimentacaoService
{
    private readonly IProdutoRepository _produtoRepository;
    private readonly IMovimentacaoRepository _movimentacaoRepository;

    1 reference
    public MovimentacaoService(IProdutoRepository produtoRepository, IMovimentacaoRepository movimentacaoRepository)
    {
        _produtoRepository = produtoRepository;
        _movimentacaoRepository = movimentacaoRepository;
    }

    1 reference
    public void RegistrarEntrada(Produto produto, Quantidade quantidade)
    {
        produto.IncluirEstoque(quantidade);

        var movimentacao = new Movimentacao(produto, quantidade, TipoMovimentacao.Entrada);
        _movimentacaoRepository.Adicionar(movimentacao);

        _produtoRepository.Atualizar(produto);
    }

    1 reference
    public void RegistrarSaida(Produto produto, Quantidade quantidade)
    {
        produto.BaixarEstoque(quantidade);

        var movimentacao = new Movimentacao(produto, quantidade, TipoMovimentacao.Saida);
        _movimentacaoRepository.Adicionar(movimentacao);

        _produtoRepository.Atualizar(produto);
    }
}
```

Princípios do padrão GRASP

Exemplos de classes no projeto que aplicam os princípios:

Creator: ProdutoFactory

Information Expert: MovimentacaoService

Low Coupling: MovimentacaoService

High Cohesion: MovimentacaoService

Controller: StockTrackApp

Pure Fabrication: ProdutoRepository, CategoriaRepository

Indirection: IProdutoRepository, IMovimentacaoRepository e ICategoriaRepository

Polymorphism: Produto > Eletronico, Alimento, Vestuario

Protected Variations: IProdutoRepository, IMovimentacaoRepository e ICategoriaRepository