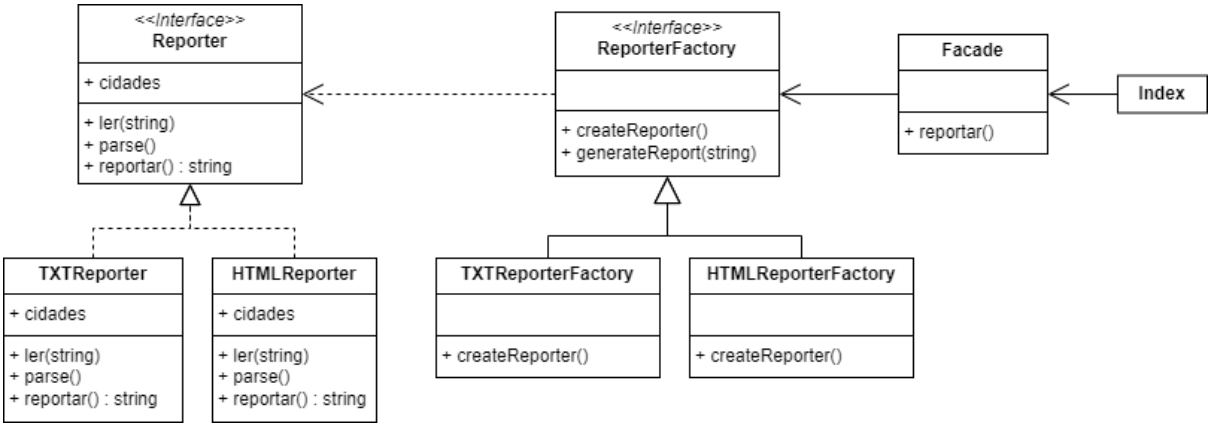
Amanda Vitória das Neves Silva
2142350

Lista 2

Padrões de projeto e refatoração

Diagrama de classes



Padrões de projeto

Facade - Estrutural

O padrão Facade é utilizado na classe *Facade* para simplificar a interface para que o cliente não precise se preocupar com os detalhes da geração do relatório. Ela possui um método *reportar()* que recebe um formato como parâmetro e retorna o relatório gerado, tornando mais fácil a utilização.

Factory Method - Criacional

O Factory Method fornece uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados.

A classe *ReporterFactory* é a base para esse padrão. Ela declara o método *createReporter()* como abstrato, deixando as subclasses responsáveis por implementar a criação do *Reporter* específico. As classes *TXTReporterFactory* e *HTMLReporterFactory* são exemplos de subclasses que implementam esse método para criar ser respectivo *Reporter*.

Amanda Vitória das Neves Silva
2142350

Template Method - Comportamental

O padrão Template Method define o esqueleto de um método ou classe, permitindo que subclasses implementem alguns de seus métodos sem mudar sua estrutura.

Ele é utilizado na classe *ReporterFactory* em conjunto com o padrão Factory Method. Nesse cenário, as classes que implementam a *ReporterFactory,* limitam-se a personalização do método *createReporter()*, enquanto o método *generateReport()* define um algoritmo geral. Em suma, parte dos métodos podem ser implementados pelas subclasses sem alterar a estrutura, mantendo sua lógica fixa.

Refatoração

Princípio da Responsabilidade Única (SRP)

O princípio da Responsabilidade Única sugere que uma classe deve ter apenas uma responsabilidade dentro do código. Garante-se a consumação desse princípio ao utilizar o padrão Factory Method. Cada subclasse de *Reporter* tem apenas uma função: gerar o relatório correspondente ao seu formato. Cada subclasse de *ReporterFactory* tem a única função de criar um *Reporter* correspondente ao seu formato. E assim por diante.

Princípio Aberto/Fechado (OCP)

O Princípio Aberto/Fechado incentiva o código a ser aberto para extensão, mas fechado para modificação. O padrão Factory Method permite adicionar novos formatos de relatório sem modificar as classes bases.

Princípio da Substituição de Liskov (LSP)

O Princípio da Substituição de Liskov diz que objetos de uma classe derivada devem ser substituíveis por objetos de sua classe base sem afetar a integridade do programa.

Caso as subclasses *HTMLReporter* e *TXTReporter* forem substituídas pela classe *Reporter*, os mesmos métodos podem ser chamados. O mesmo serve para a *ReporterFactory* com as subclasses *HTMLReporterFactory* e *TXTReporterFactory*.

Amanda Vitória das Neves Silva
2142350

Princípio da Segregação de Interface (ISP)

O Princípio da Segregação de Interface sugere que uma classe não deve ser forçada a depender de interfaces que não utiliza.

A utilização do Factory Method na classe *Reporter* define explicitamente o método que cada implementação deve executar, sendo específica e concisa. A combinação do Factory Method com o Template Method na classe *ReporterFactory* deixa explícito qual o método abstrato a ser implementado pelas subclasses e quais métodos já estão implementados seguindo as regras de negócio.

Princípio da Inversão de Dependência (DIP)

A inversão de dependência entre as classes *ReportFactory* (nível mais alto) e *Report* (nível mais baixo) permite que a *ReportFactory* trabalhe com qualquer classe que implemente a interface *Report*, tornando o sistema mais flexível e aberto a extensões.

Implementação

O projeto implementado pode ser acessado através deste link. Mais informações quando a execução do projeto estarão descritas no arquivo readme do repositório.

O código base para refatoração e implementação de padrões pode ser acessado através deste link.