UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB
CURSO: BACHARELADO EM SISTEMAS DE INFORMAÇÃO
DISCIPLINA: PROJETOS DE SISTEMAS DE INFORMAÇÃO II
DOCENTE: ALCEMIR SANTOS
DISCENTES: DOUGLAS AUGUSTO, JOÃO RAMYLLO, RENAN RÊGO, SALES MACÊDO

PADRÕES DE PROJETOS UTILIZADOS NO SISTEMA DE GERENCIAMENTO E ARMAZÉM DE SEMENTES

INTRODUÇÃO

Os *Design Patterns* (Padrões de Projetos) têm sua origem no trabalho de um arquiteto chamado Christopher Alexander em meados da década de 70. Ele escreveu dois livros de bastante sucesso que exemplificava o uso e descrevia seu raciocínio para documentar os padrões para a arquitetura. Em 1995, um grupo de quatro profissionais (que ficou conhecido como *Group Of Four* ou Grupo dos Quatro) escreveu e lançou o livro *"Design Patterns: Elements of Reusable Object-Oriented Software"* [Gamma95] que continha um catálogo com 23 padrões de projetos orientados a software. A ideia de documentar problemas recorrentes que acontecia nos *softwares* surgiu através da ideia de Christopher Alexander que também percebeu essa necessidade na sua área.

Os *Design Patterns* são uma coleção de padrões de projeto de software que contém soluções para problemas conhecidos e recorrentes no desenvolvimento de software descrevendo uma solução comprovada para um problema de projeto recorrente. A Documentação desses padrões permite o reuso e o compartilhamento dessas informações sobre a melhor maneira de se resolver um problema de projeto de *software*.

Em seguida, serão documentados os padrões que foram utilizados no projeto.

1. FACTORY METHOD

Como todo padrão de criação *Factory*, este padrão encapsula a criação de objetos. A diferença é que encapsula-se a criação de objetos deixando as subclasses decidirem quais objetos criar.

No projeto, foi criado um pacote com as classes Admin, Funcionário e PessoaFactory e a Interface UsuarioIF, as quais as demais classes implementa o método *criarPessoa*. A classe PessoaFactory é uma fábrica de objetos e decide qual tipo de objeto será criado dependendo de apenas um parâmetro que lhe é passado.

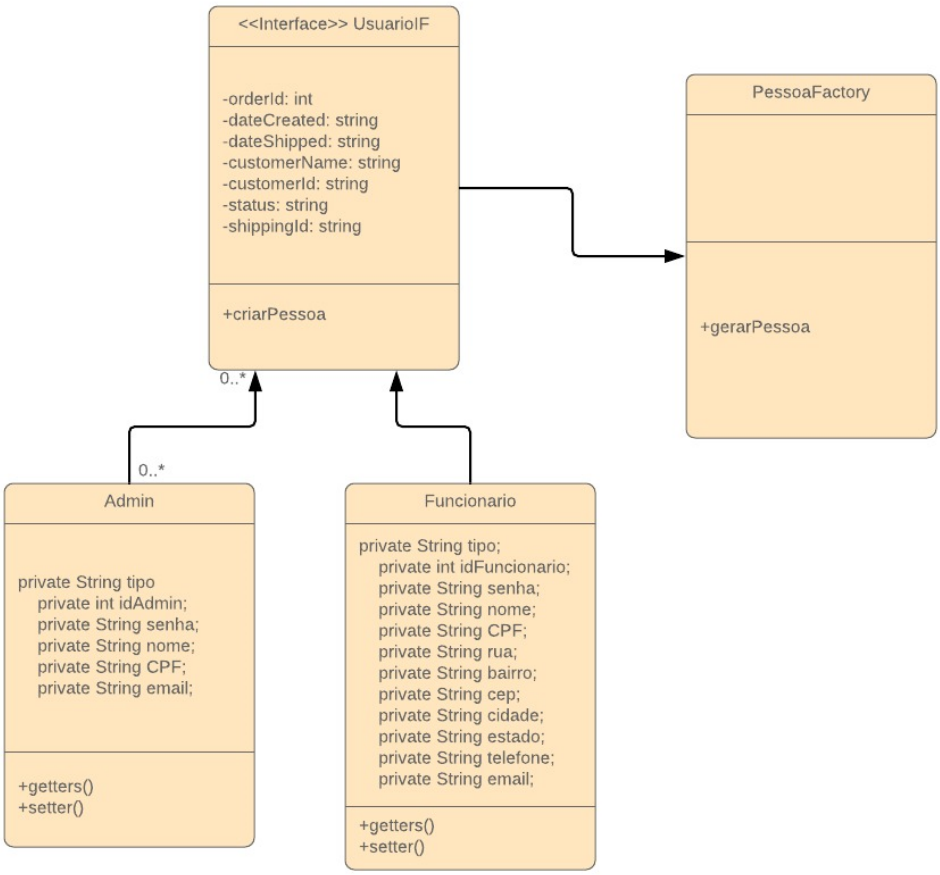


Figura 1 - Diagrama de classes - FACTORY METHOD

2. SINGLETON

Sua utilização se dá quando se deseja que a aplicação possua apenas uma instância de uma determinada classe. Isso normalmente é motivado por questões de negócio, em que faz sentido apenas um objeto daquele tipo. Por exemplo, imagine um software que represente um jogo de xadrez entre duas pessoas. Nesse contexto provavelmente fará sentido apenas um tabuleiro de jogo. A estrutura para permitir esse tipo de construção é o padrão Singleton .

A listagem a seguir apresenta um exemplo de implementação do padrão Singleton . No exemplo, o Singleton é utilizado em uma classe que representa e armazena configurações do sistema. Esse é um exemplo em que o uso desse padrão é adequado, pois só existe uma única configuração para todo o sistema, e dessa forma ela pode ser facilmente obtida a partir de qualquer classe.

No projeto, foi criado a classe MakeConnectionSingleton2, que oferece os métodos para criação de conexões com o Banco de Dados de forma única, através de uma única instância. O que garantirá essa unicidade da instância única é a variável *instancy* privada e estática do tipo MakeConnectionSingleton2. Essa variável será alocada no método *getInstancy*, declarado na mesma classe. Se a variável já tiver sido instanciada, ela retornará a própria instância em si. O método getConexao retornara a conexão instanciada anteriormente para as classes que utilizarem a conexão realizada.

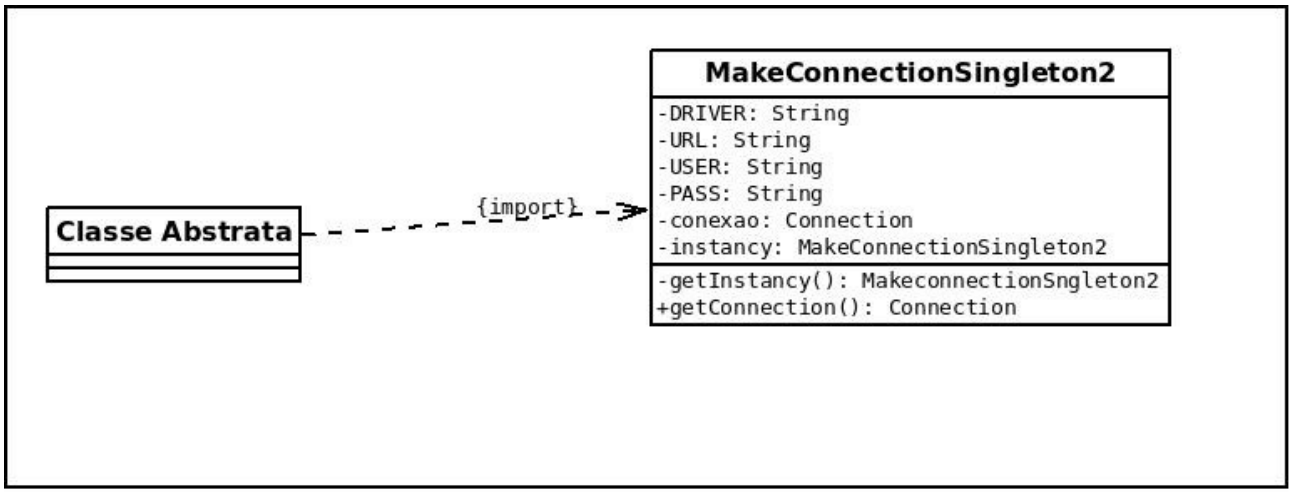


Figura 2 - Diagrama de classes - SINGLETON

3. STRATEGY

Categorizado como um padrão comportamental de desenvolvimento de software, o *Strategy* permite definir novas operações sem alterar as classes dos elementos sobre os quais opera. Segundo o catálogo GOF o padrão tem como meta: "Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis.

O maior incentivo para uso do padrão é a melhoria da manutenção do código. Permite trocar o algoritmo de uma classe dinamicamente, ou seja, minimiza a quantidade if/else desnecessário para executar uma lógica baseada em um tipo.

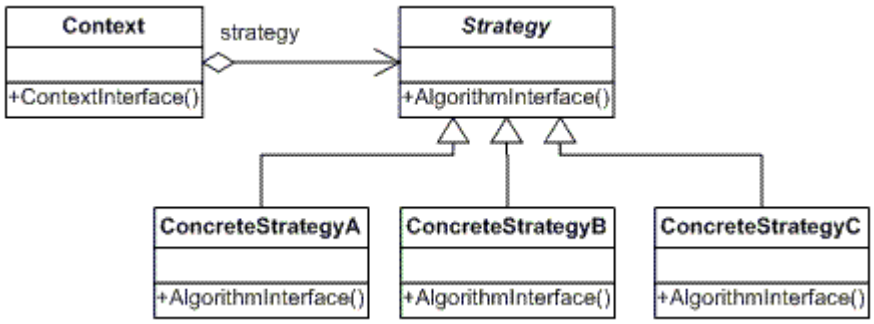
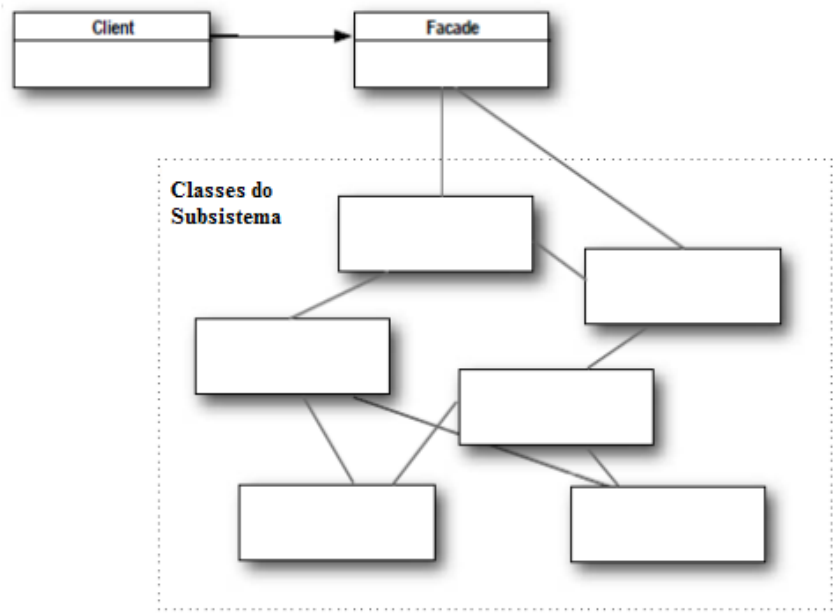


Figura 3 - Diagrama de classe do Strategy

No projeto, foi criado um pacote de validação de datas de colheita e validade com o intuito de passar datas válidas nos casos de testes do projeto. Foram criadas uma interface ValidaDataSemente e mais três classes representando o dia, mês e ano. Cada classe implementa o método da interface e cada uma é especializada em retornar um valor booleano. Esta é uma boa prática de programação, pois minimiza a quantidade de if/elses, facilitando assim, a manutenibilidade do código.

4. FACADE

O Padrão de Projeto Facade oculta toda a complexidade de uma ou mais classes através de uma Facade (Fachada). A intenção desse Padrão de Projeto é simplificar uma interface. Com o Padrão Facade podemos simplificar a utilização de um subsistema complexo apenas implementando uma classe que fornece uma interface única e mais razoável, porém se desejássemos acessar as funcionalidades de baixo nível do sistema isso seria perfeitamente possível. É importante ressaltar que o padrão Facade não "encapsula" as interfaces do sistema, o padrão Facade apenas fornece uma interface simplificada para acessar as suas funcionalidades.



No projeto, foi criada a classe Facade, contendo todas as instâncias das classes e os métodos que as utilizam e em cada cadastro e alterações das classes há uma instância de Facade para acessar as instâncias e os métodos as quais a fachada conhece.

5. DAO (Data Access Object)

O padrão de projeto DAO surgiu com a necessidade de separarmos a lógica de negócios da lógica de persistência de dados. Este padrão permite que possamos mudar a forma de persistência sem que isso influencie em nada na lógica de negócio, além de tornar nossas classes mais legíveis. Classes DAO são responsáveis por trocar informações com o SGBD e fornecer operações CRUD e de pesquisas, elas devem ser capazes de buscar dados no banco e transformar esses em objetos ou lista de objetos, fazendo uso de listas genéricas, também deverão receber os objetos, converter em instruções SQL e mandar para o banco de dados.

Toda interação com a base de dados se dará através destas classes, nunca das classes de negócio, muito menos de formulários. Se aplicarmos este padrão corretamente ele vai abstrair completamente o modo de busca e gravação dos dados, tornando isso transparente para aplicação e facilitando muito na hora de fazermos manutenção na aplicação ou migrarmos de banco de dados.

Também conseguimos centralizar a troca de dados com o SGBD (Sistema Gerenciador de Banco de Dados), teremos um ponto único de acesso a dados, tendo assim nossa aplicação um ótimo design orientado a objeto.

No projeto, o padrão foi aplicado no pacote DAO, responsabilizando às suas classes, a função da comunicação com o BD, referente ás classes que as utilizam. Sendo elas: AdminDAO, FornecedorDAO, FuncionarioDAO, SementeDAO e VendaDAO.