Atividade de Avaliação do Módulo de Refatoração

Curso de Especialização Desenvolvimento Full-Stack - Faculdade Delta

Flávio de Souza

1. Objetivo da atividade:

Dentro do seu ambiente de trabalho, encontrar 4 situações que podem ser aplicados, todos os padrões de projeto (*Strategy*, *Decorator*, *Abstract Factory* e *Observer*), conforme *script* descrito a seguir:

- Explicar o Padrão de Projeto detalhadamente
- Elaborar o Diagrama de Classes do cenário aplicado o *pattern*
- Explicar o benefício da adoção do *pattern* no cenário de código em questão.

2. Descrição das situações a serem abordadas

Para o cumprimento dos requisitos do presente trabalho, todas a situações levantadas estão relacionadas ao contexto de funcionalidades associadas ao processamento e interpretação de arquivos de remessa de títulos de cobrança bancária.

É muito comum que no relacionamento com seus clientes as empresas disponibilizem boletos como forma de pagamento por produtos e serviços oferecidos. Para a emissão e recebimento de por boletos – à grosso modo – é preciso trocar informações com as instituições bancárias, tanto para enviar os títulos de cobranças gerados, como para receber a atualização da situação dos mesmos, como por exemplo o pagamento dos mesmos por parte do cliente.

Para tal, existem padrões de arquivos associados a esses tipos de transações, onde existem arquivos específicos de remessa de títulos de cobrança, sendo que cada arquivo contém diferentes tipos de registros. Por exemplo, um arquivo de remessa de envio de títulos para o banco possui um registro de cabeçalho contendo a informação do cedente e do convênio associados à remessa, juntamente com outros diferentes tipos de registros com informações distintas.

O presente trabalho apresentará padrões de projetos empregados para resolver problemas relacionados à cenários inseridos nesse contexto. Sendo que, existe um sistema em produção que implementa essa solução, porém, para atender aos requisitos solicitados, foi desenvolvida uma solução dentro de um escopo reduzido que atende a este trabalho, o qual está disponível em: https://github.com/flaviodev/refactoring

3 . Aplicação de padrões de projeto no processo de refatoração

3.1. Strategy

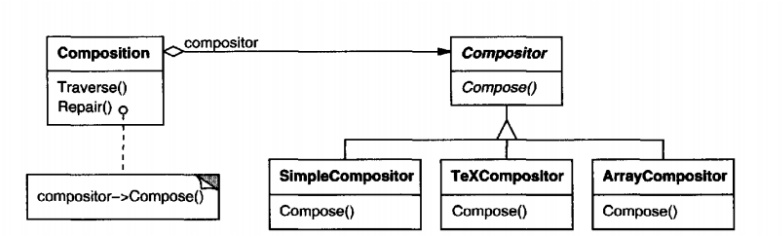
Trata-se de uma família de algoritmos, que encapsula diferentes comportamentos tornando-os intercambiáveis. Permitindo assim, que um comportamento (algoritmo) varie independentemente dos clientes que o usam, possibilitando deste modo uma implementação com baixo nível de acoplamento.

A base desse padrão se concentra no uso do polimorfismo (seja por meio de uma classe abstrata ou por interface) onde o cliente, onde o cliente invoca um determinado comportamento, sem no entanto, ter a necessidade de conhecer a sua implementação.

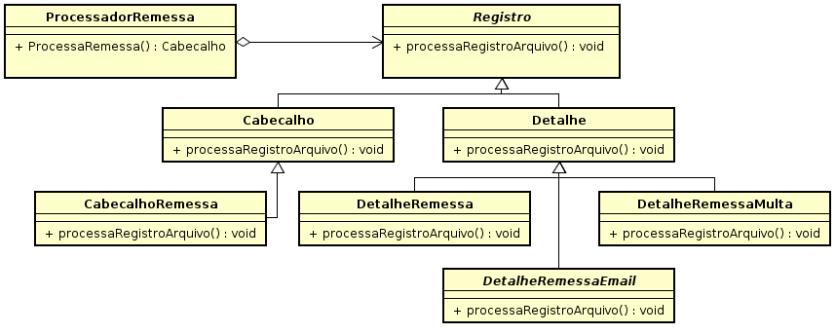
Clientes que precisam de implementação de tratamentos condicionais tornam-se – naturalmente – mais complexos, dificultando assim a manutenibilidade e testabilidade do código. Pois além do elevado grau de complexidade ciclomática do código, a cada novo tratamento – a cada nova variação – exige-se uma refatoração no código do lado do cliente.

Esse padrão permite que os comportamentos variem em detrimento do cliente, ou seja, o comportamento do cliente não muda quando se deseja implementar uma nova variação de um determinado comportamento. Emilinando assim *enormes switches e cadeias de if-else no lado do cliente.*

Modelo implementação:



Implementação no Cenário:



Benefício da adoção no cenário:

A solução possibilitou o desacoplamento das regras de negócio de leitura de cada registro em uma classe distinta no código, uma vez que cada linha do arquivo de remessa possui um layout distinto conforme o tipo, com atributos diferentes e em posições na linha do registro.

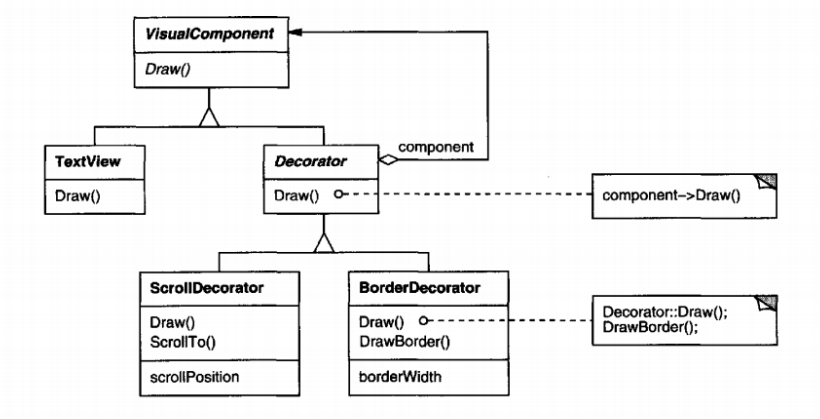
Outro grande ponto importante é que existem diferentes tipos de layouts de remessa, bem como diferentes tipos de registros, o padrão strategy permite a implementação gradativa dos layouts – considerando que podem haver diferenciações para cada banco – sem o processador de remessas seja alterado. Por exemplo, a presente implementação contempla somente os arquivos de remessa, sendo que a implementação do processamento da remessa de retorno não implicaria em refatoração do processador de remessas (cliente).

3.2. Decorator

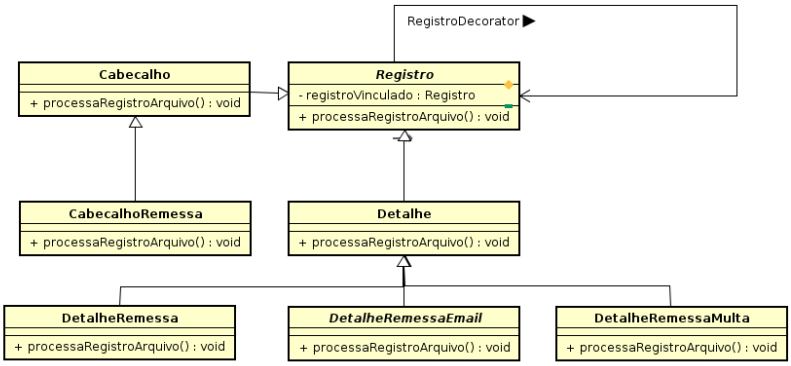
Anexa responsabilidades adicionais a um objeto de forma dinâmica, este padrão permite que uma classe estenda uma funcionalidade de forma flexível. É um meio muito útil de compor – combinar – comportamentos de forma flexível e dinámica de uma classe.

O princípio básico está na implementação e uma referência de um comportamento adicional ao objeto, que ao ser criado pode receber – ou não – um comportanto (que “decora-o”) adicionando um novo comportamento (sendo que novo comportamento pode vir acompanhado de um um outro decorator, o que permite a adição “ilimitada” de decorators ao objeto.

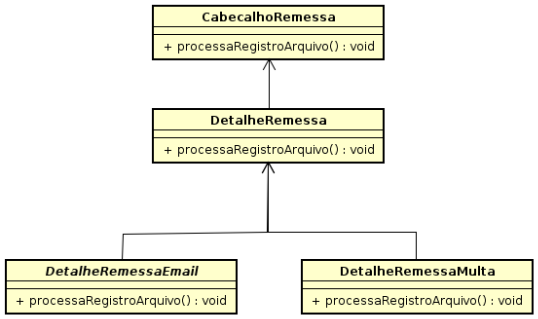
Modelo implementação (GoF):



Implementação no Cenário:



** implementação conforme curso de design patterns da alura, onde a própria interface de Registro atua como Decorator, passando o decorador como argumento no construtor*



*** Resultado da aplicação do padrão decorator onde um registro pode ser vinculado a outro registro de forma dinámica*

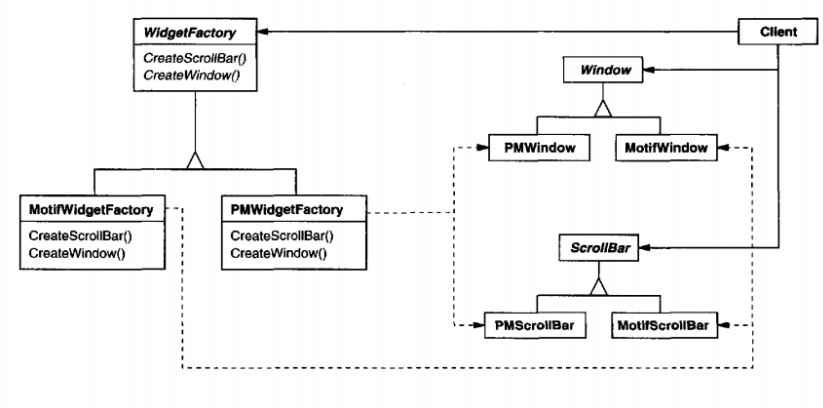
Benefício da adoção no cenário:

Como muitos regitros dos arquivos de remessa se relacionam entre si, a aplicação do padrão de projeto permitiu (de forma simples) por meio da vinculação de vários registro a construção de um objeto de TítuloCobranca. Bem como, possibilita a implementação de outros layouts de remessa sem a necessidade de refatorar a rotina de processamento de remessas. Como por exemplo, é possível implementar o processamento dos arquivos de retorno do banco sem alterar o código implementado para o processamento dos arquivos de remessa de envio.

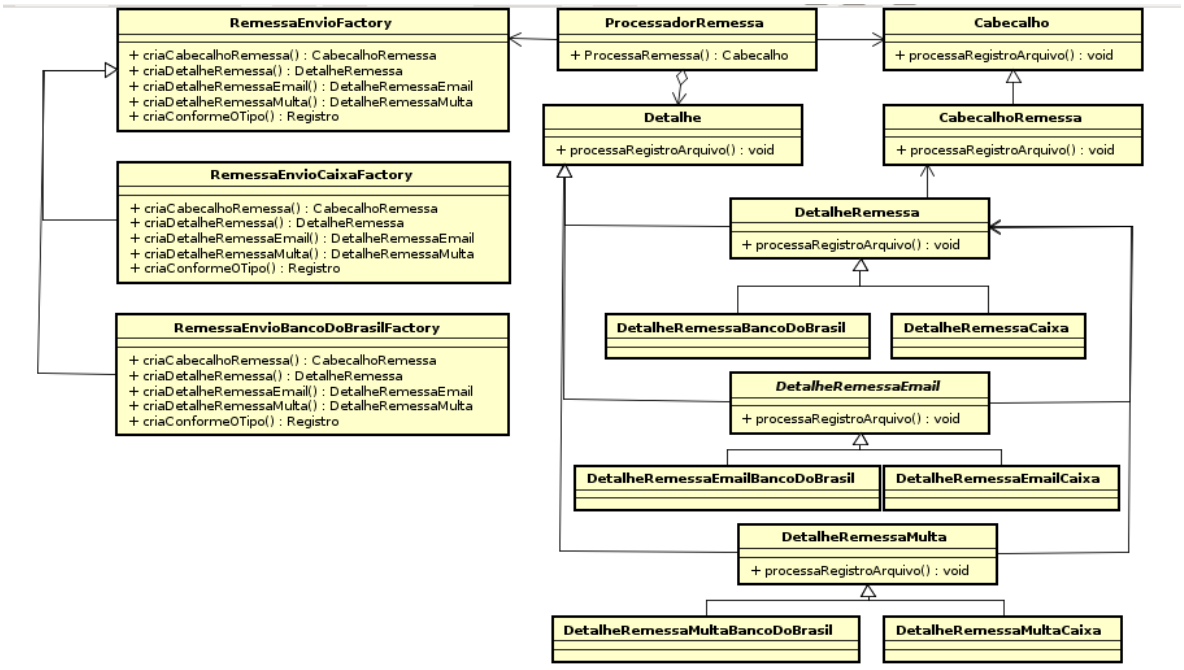
3.3. Abstract Factory

Fornece uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Possuindo um comportamento semelhante ao strategy, contudo sendo reforçado o comportamento de “família de objetos”, seria equivalente ao um strategy de um conjunto de objetos.

Modelo implementação:



Implementação no Cenário:



Benefício da adoção no cenário:

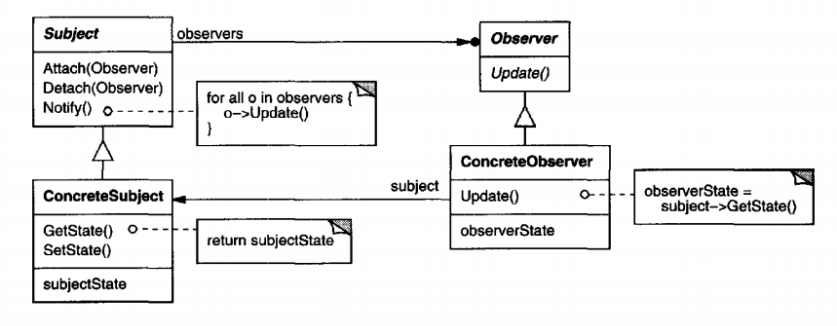
Seguindo a mesma linha dos benefícios das implementações dos padrões anteriores, o principal ponto está relacionado à evolução da solução sem necessidade refatoração no núcleo da funcionalidade (processamento das remessas). O abstract factory permitiu aqui a estensibilidade da solução para qualquer banco, pois mesmo com o padrão instituído pela federação dos bancos, há diferenças na implementação conforme o banco (exemplo registro de retorno do banco para a alteração do cnpj de um título).Há uma especialização de cada banco para cada tipo de registro, o abstract factory permite obter esse o conjunto de implementaçãoes para cada banco na hora de processar um arquivo. A parte mais importante está no de que é possível acrescentar de forma relativamente fácil outros bancos

(implementando somente os tratamentos particulares), sem interferir no código das rotinas dos bancos já implementados.

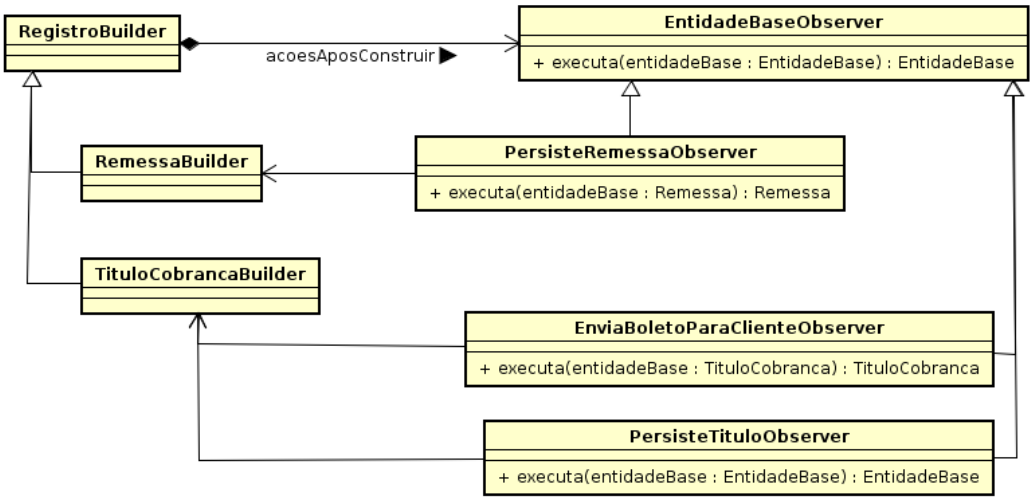
3.4. Observer

Determina uma dependência do tipo “um para muitos” entre objetos, onde, quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados de forma automática, ou seja – como o próprio nome diz – define observadores ao estado de um objeto. Este padrão permite implementar comportamentos a um objeto de forma dinâmica e com baixo acoplamento e alta coesão, possibilitando assim que o design da classe se preocupe exclusivamente com suas responsabilidades.

Modelo implementação:



Implementação no Cenário:



Benefício da adoção no cenário:

A adoção de padrões de projetos estão diretamente ligadas às boas práticas das orientação à objeto, onde sempre é desejável um baixo acoplamento e alta coesão. Aqui o observer permitiu adicionar de forma simples e flexível e simples comportamentos à construção de cada registro do layout da remessa. Possibilitando assim poder escolher o que fazer e a que momento fazer uma operação no processamento de um registro da remessa. Efetuar ações distintas é muito comum, pois em um dado momento é desejado processar o registro e persistir em banco de dados, em outro momento é necessário gerar um arquivo texto, ou gerar um boleto, e etc. O mais importante é que com essa solução novos comportamentos podem ser acrescentados a qualquer momento com impácto muito baixo na implementação do processo principal da funcionalidade.