**INSTUTO FEDERAL DO ESPÍRITO SANTO**

BRUNO DELL’ORTO MERGH

GABRIELA PONTES BREDER

**TRABALHO DE PROGRAMAÇÃO ORIENTADA A OBJETOS 2**

SERRA

2015

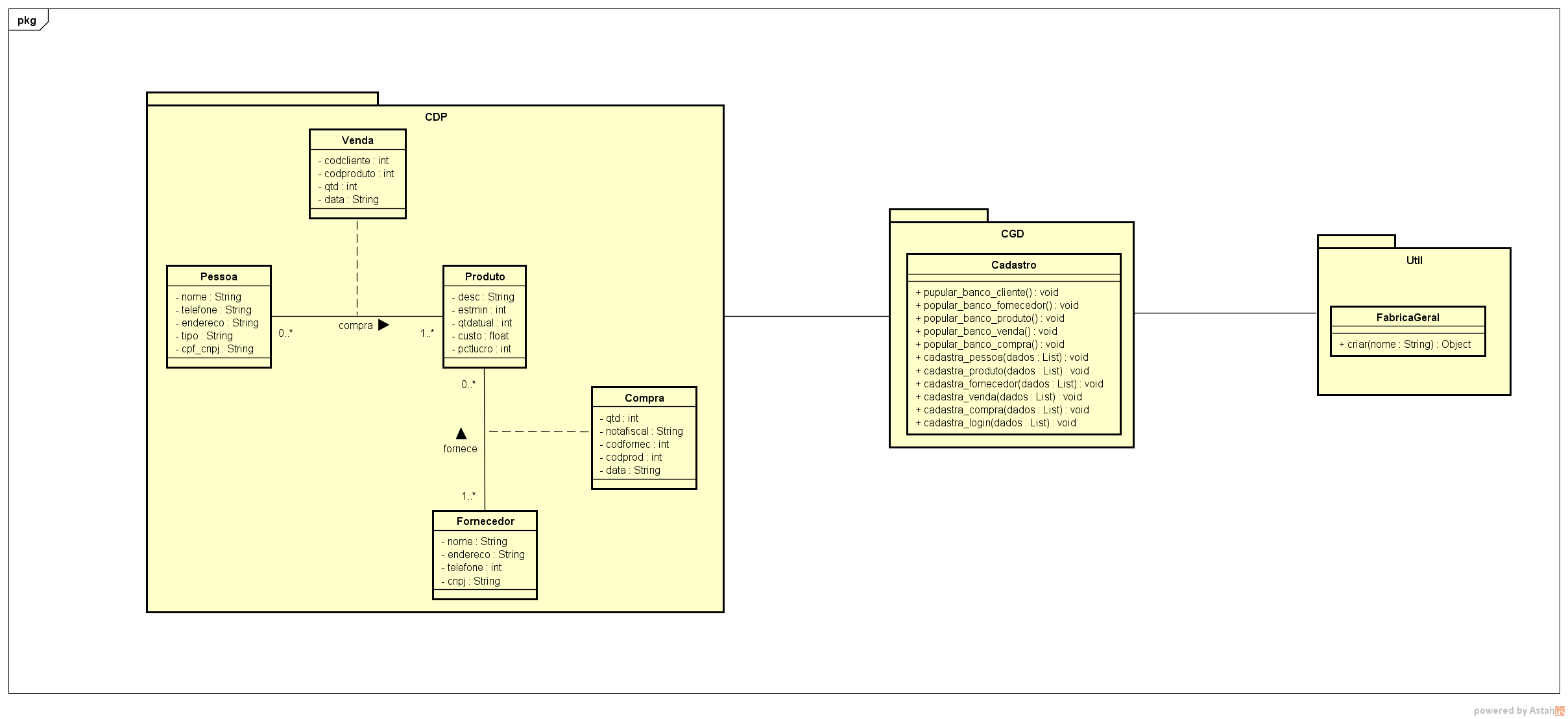
**Sumário**

Explicação do uso e seus respectivos diagramas de padrões de projeto..........................Pág. 2

Descrição de utilização do padrão MVC.............................................................................Pág. 9

Refatoração e avaliação da qualidade do código...............................................................Pág. 12

**EXPLICAÇÃO DO USO E SEUS RESPECTIVOS DIAGRAMAS DE PADRÕES DE PROJETO**

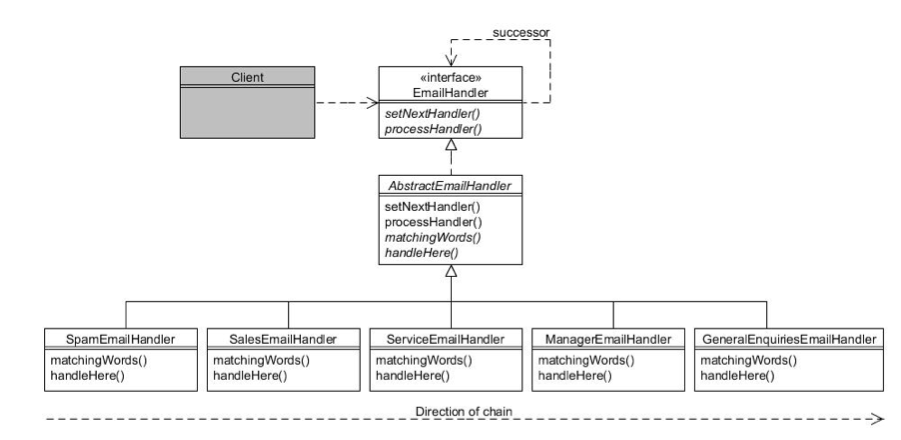
**Visão sem os novos diagramas:**

Neste trabalho iremos abordar os padrões de projeto chamados de comportamentais. São os padrões que estão preocupados com os algoritmos e as atribuições de responsabilidade entre objetos. Descrevem não só os padrões entre objetos ou classes, mas também os padrões de comunicação entre eles. Estes padrões caracterizam um complexo fluxo de controle que é difícil de seguir em tempo de execução. Eles transportam sua atenção para longe do fluxo de controle e lhe permite concentrar se apenas no modo como os objetos estão interconectados.

Lista de padrões comportamentais:

1. Cadeia de responsabilidade

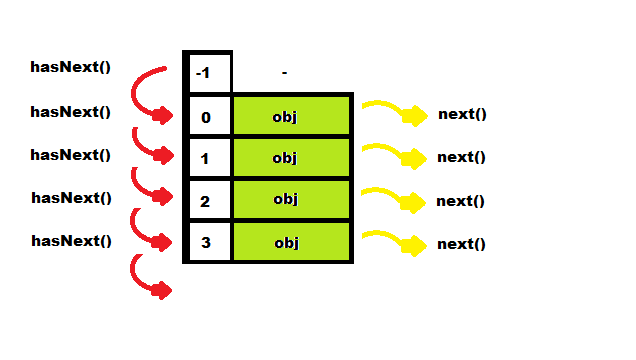
O padrão de projeto de software representa um encadeamento de objetos receptores para o processamento de uma série de solicitações diferentes. Esses objetos receptores passam a solicitação ao longo da cadeia até que um ou vários objetos a tratem.



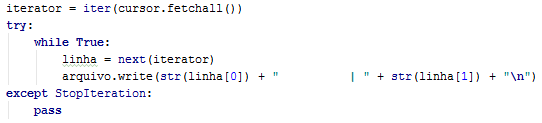
Como visto em sala de aula e no exemplo acima, este padrão não se encaixa em nosso minimundo, pois não temos tratamentos diferenciados de um mesmo objeto.

1. Iterator

O Padrão de Projeto Iterator tem como objetivo encapsular uma iteração.

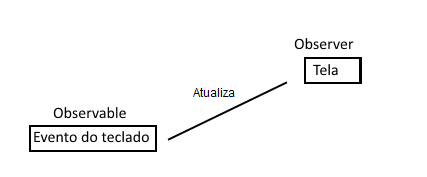


Este padrão foi aplicado na classe "relatorio.py" em todas seus métodos, estes métodos por sua vez tem o objetivo de acessar o banco de dados e guardar em uma lista o seu objetivo. Após termos esta lista, criamos a chamada do iterator para que ele termine o processo e armazene no arquivo de destino.

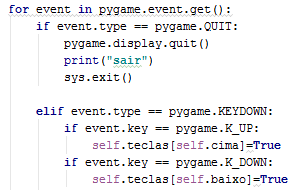


1. Observador

Este padrão define a dependência de um objeto perante outros objetos e, assim, quando a estado de um objeto é mudado, os objetos dependentes são notificados e atualizados automaticamente.

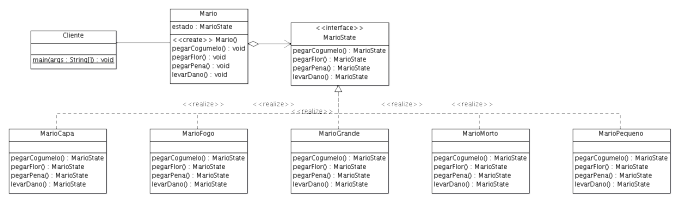


Em nosso trabalho, o padrão observador foi aplicado para controlar os eventos do teclado, como visto na figura acima, o observador analiza o teclado e após o uso ele atualiza um determinado objeto na tela.



1. Estado

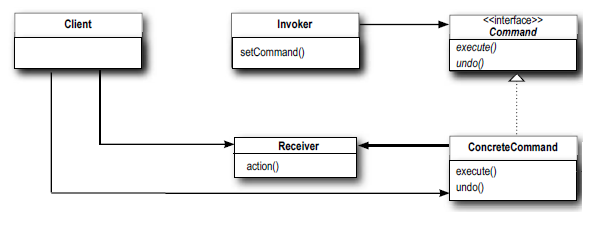
Este padrão permite que o objeto mude o seu comportamento, conforme o seu estado. Por exemplo, em jogos podem existir itens que alteram o mesmo personagem, como no jogo muito conhecido, Super Mario World, o mesmo personagem durante a "fase" pode mudar seu comportamento (voar, diminuir de tamanho, crescer, atirar fogo,etc.).



Em nosso projeto, infelizmente, não temos a possibilidade de aplicar este padrão,conforme explicado anteriormente.

1. Comando

O padrão comando nos diz como criar “objetos de comando” que encapsula uma solicitação para fazer algo em um objeto específico. Basicamente todos os objetos de Comando implementam a mesma interface, que consiste em um único método normalmente chamado de execute(). Ele também tem como definição encapsular uma solicitação como um objeto, o que lhe permite parametrizar outros objetos com diferentes solicitações, enfileirar ou registrar solicitações e implementar recursos de cancelamento de operações.

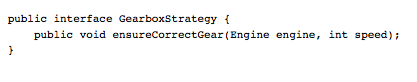


No diagrama de classe acima tem-se o Client que é quem cria um Comando concreto e quem define o receptor. Em nosso minimundo pensamos em aplicar o padrão comando para informar quando tempo falta para expirar o uso do nosso software, e assim o tornando inviável a sua utilização depois de uma certa data. O comando em si faria com que o usuário não conseguisse passar da tela de login enquanto o contrato não fosse renovado.

1. Estratégia

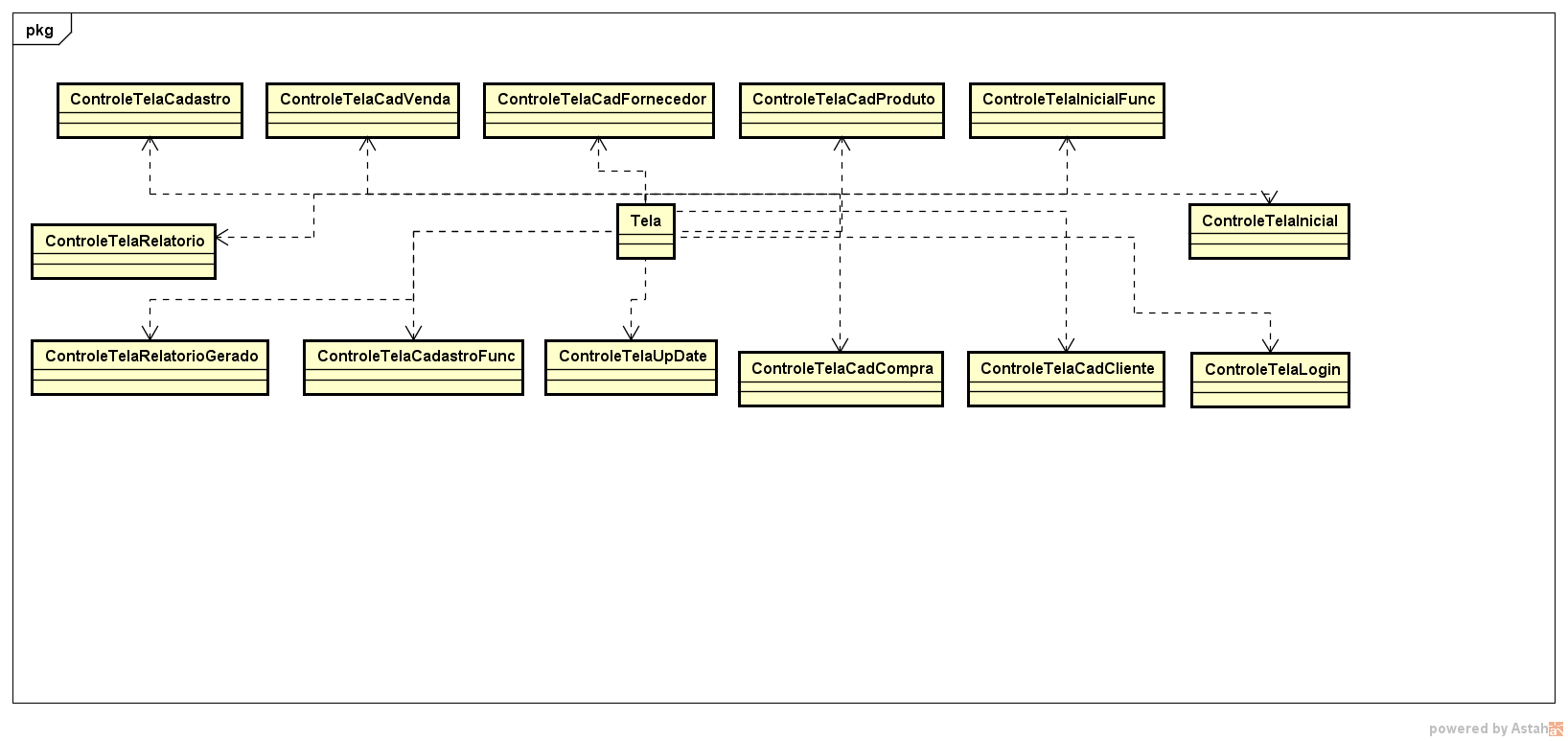
Este padrão permite que o cliente mude o algoritmo de um objeto que utiliza uma função. Como por exemplo, em nosso minimundo podemos adicionar um pacote promocional na classe produto, para que todo cliente que pagar sua fatura no dia 25 de cada mês receba um desconto de 5% no valor total dos produtos. Para isso será preciso criar um método chamado *boxStrategy* passando para ele o dia atual e o valor de cada cliente, se o dia do pagamento for igual a 25, a *boxStrategy* retorna o valor com desconto.

Abaixo segue um exemplo ensinado em sala de aula.



1. Mediador

Este padrão define um objeto que encapsula as interação de um conjunto de projetos, em outras palavras, levando pro lado do nosso projeto, este padrão tem a responsabilidade de guardar com ele todas as chamadas dos respectivos métodos usados no corpo da função, sendo esta, responsável por fazer a chamada e distribuir as informações para os outros controles dependendo da opção desejada.



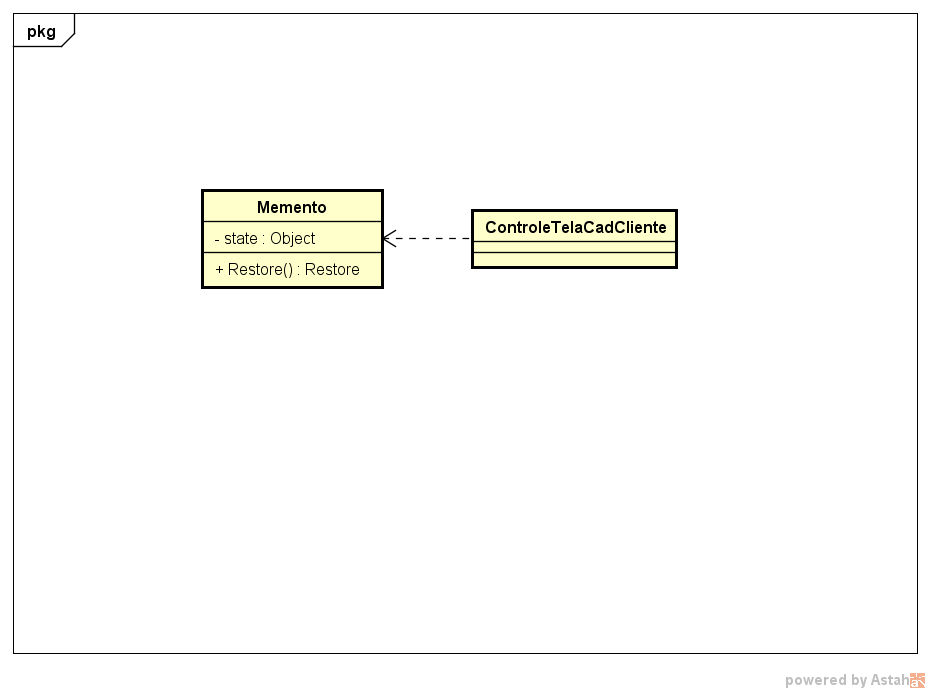
1. Template Method

Um Template Method auxilia na definição de um [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) com partes do mesmo definidos por [métodos abstratos](https://pt.wikipedia.org/wiki/M%C3%A9todo_abstrato). As sub[classes](https://pt.wikipedia.org/wiki/Classe) devem se responsabilizar por estas partes [abstratas](https://pt.wikipedia.org/wiki/Abstrata), deste [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo), que serão implementadas, possivelmente de várias [formas](https://pt.wikipedia.org/wiki/Formas), ou seja, cada [subclasse](https://pt.wikipedia.org/wiki/Subclasse) irá implementar à sua necessidade e oferecer um comportamento [concreto](https://pt.wikipedia.org/wiki/Concreto) construindo todo o [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo).

Em nosso trabalho anteriormente existia a classe abstrata *Pessoa*, porém após diversas atualizações resolvemos remover a classe abstrata *Pessoa* colocando em um dos seus atributos a tipo de pessoa, física ou jurídica.

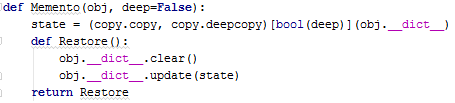
1. Memento

Uma explicação rápida e fácil de entender desse padrão foi nos dada em sala de aula, o padrão captura o estado de um objeto sem violar o encapsulamento e, dessa forma, prover um meio de restaurar o estado inicial do objeto.

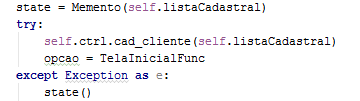


Em nosso trabalho foi aplicado este padrão no momento em que cadastramos os produtos, caso ocorra alguma exceção, o M*emento* salva o ultimo estado e mantém a tela fixa com os dados.

Classe:

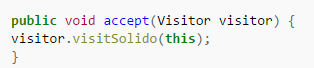
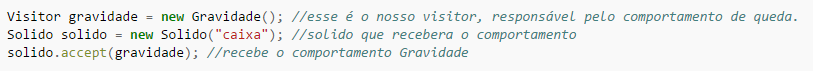


No código:



1. Visitador

O Visitador permite que se crie uma nova operação sem que se mude a classe dos elementos sobre as quais ela opera. É uma maneira de separar um [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) da estrutura de um objeto. Um resultado prático é a habilidade de adicionar novas funcionalidades a estruturas de um objeto pré-existente sem a necessidade de modificá-las.

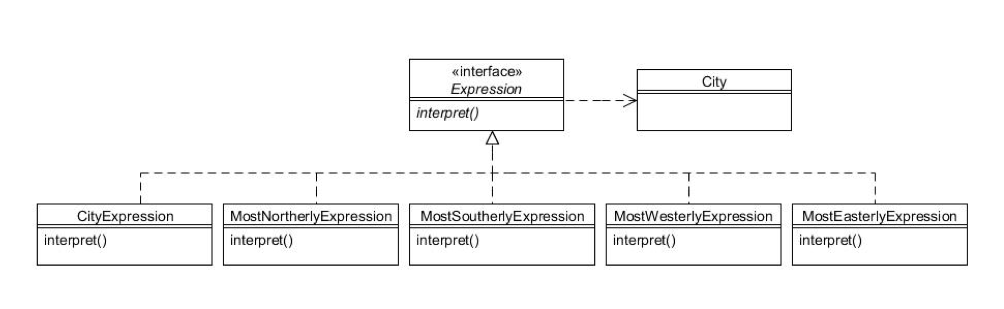


Ao passar para o Visitador uma referencia de si mesmo, o visitor pode acessar os métodos e atributos públicos dessa classe, que no nosso caso, vai adicionar a aceleração da gravidade ao Solido. Assim como o comportamento de queda foi adicionado, outros também poderiam ser feitos da mesma maneira, como movimentação através do teclado, sons... as possibilidades são infinitas.

Depois de muitas leituras e exemplos, não vi a possibilidade de encaixar este padrão em nosso trabalho.

1. Interpreter

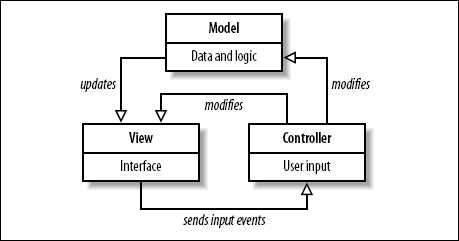
Este padrão por sua vez define a representação de uma gramática, por exemplo, o sistema permite ao usuário perguntar qual cidade está mais ao sul, norte, leste ou oeste da sua posição. Abaixo segue o diagrama do exemplo:



Padrão não implementado por não se adaptar a realidade do projeto.

**DESCRIÇÃO DA UTILIZAÇÃO DO MVC**

O padrão Modelo-Visão-Controlador (MVC) considera três papéis relacionados à interação humano-computador. O modelo consiste nos dados da aplicação, regras de negócios, lógica e funções. A visão refere-se à entrada e a exibição de informações na IU. Pode ser qualquer saída de representação de dados. Qualquer requisição é tratada pelo terceiro papel: o controlador. Este pega a entrada do usuário, envia uma requisição para a camada de lógica de negócio, receber sua resposta e solicita que a visão se atualize conforme apropriado. É então a mediação da entrada, convertendo-a em comandos para o modelo ou visão.



* CDP

Na Camada de Domínio do Problema estão as classes *compra*, *fornecedor*, *pessoa*, *produto* e *venda*. Todas essas classes são identificadas na fase de análise. As classes pfisica e pjuridica da parte 1 do trabalho foram retiradas, pois ao adicionarmos o uso do banco de dados colocamos um atributo tipo na classe pessoa. Além disso, criamos uma nova classe login, utilizada por conta da interface.

* CGD

Na Camada de Gerência de Dados estão as classes cadastro e relatorio. Como o nome diz, essa camada compreende a definição das classes gerenciadoras de tarefas. Na classe *cadastro*, estão as funções de popular o banco, que fazem a leitura/backup dos arquivos existentes. Além disso, há as funções de cadastro de pessoa, produto, fornecedor, venda e compra. Já na classe *relatorio* são gerados os arquivos WriteApagar (valor devido aos fornecedores), WriteAreceber (valor a receber por cliente), WriteVendasPorProduto (vendas e lucro por produto) e WriteEstoque (gerenciamento de todos os produtos do estoque). Nesta camada então estão as classes que requerem o armazenamento de dados.

A descrição acima já estava implementada na parte 1. O que mudou agora foi a adição das classes DAO, relacionadas ao uso do banco de dados: *daocliente*, *daocompra*, *daofornecedor*, *daologin*, *daoproduto* e *daovenda*. Elas se conectam ao banco e possuem as funções de inserir, retornar (cliente, compra, fornecedor...), e deletar. Outra mudança foi a criação da classe facade, para a implementação desse padrão de projeto.

* CIH

O Componente de Interação Humana trata do projeto da interação humano-computador. Essa camada foi a que mais sofreu modificações, por conta da criação de uma interface. Antigamente, possuía somente a classe menu, que imprimia as opções de entrada ao cliente e estabelecia então a comunicação entre o usuário e o computador. Foram criadas no CIH classes que exibem as telas de cadastro, tela inicial, de login, relatórios, enfim, tudo que a interface exibe ao usuário.

* CCI

O Componente de Controle de Interação trata das classes responsáveis por controlar a interação (ativação/desativação dos objetos do CIH). Na parte 1, o CCI possuía apenas a classe *control* - que, dado o valor do input do usuário, chamava as classes e funções responsáveis para executar o que ele desejava.

Nesta camada também foram adicionadas muitas outras classes em decorrência das telas. Para cada uma delas foi criada uma classe de controle respectiva que faz basicamente o mesmo que a *control*, porém agora através da interface. Por exemplo, se na tela de cadastro o usuário aperta enter na opção cliente, a classe *controleTelaCadastro* do CCIchama a classe *TelaCliente* que está no CIH e a partir daí o usuário será redirecionado para a tela de cadastro do cliente e poderá fazer o que deseja.

* UTIL

No pacote Util fizemos uma modificação nas classes do Método Fábrica. Na primeira parte, o código possuía uma fábrica para cada classe do nosso diagrama (*pessoa* – *fabricaPessoa*, *produto* – *fabricaProduto*...). Refatorando o código, deixamos apenas uma classe *fabricaGeral* que possui o método criar e a *flyweightFactory* para a aplicação do padrão Flyweight. Nessa ultima atualização foi criado também a classe Memento, para encaixar nos padrões estipulados no trabalho.

* APPLICATION

Já no pacote Application está a classe *main*, que cria os objetos DAO e as tabelas necessárias. A partir do main, há o gerenciamento das telas e todo o programa é executado.

* ARQUIVOS

Neste pacote estão todos os arquivos de entrada necessários caso o usuário queira fazer o backup do banco de dados. Além deles, os relatórios gerados também são salvos neste pacote.

* TESTE

Esta classe, como o nome já diz, é para aplicar o TDD (Desenvolvimento Orientado a Testes). Para cada funcionalidade do sistema é criado um teste antes. Essa boa prática garante um software com código mais limpo e coeso. Em nosso trabalho, por exemplo, fazemos testes nos atributos de pessoa, produto, fornecedor etc e também para testar o resultado dos relatórios. Testes continuam os mesmos.

**REFATORAÇÃO E AVALIAÇÃO DA QUALIDADE DO CÓDIGO**