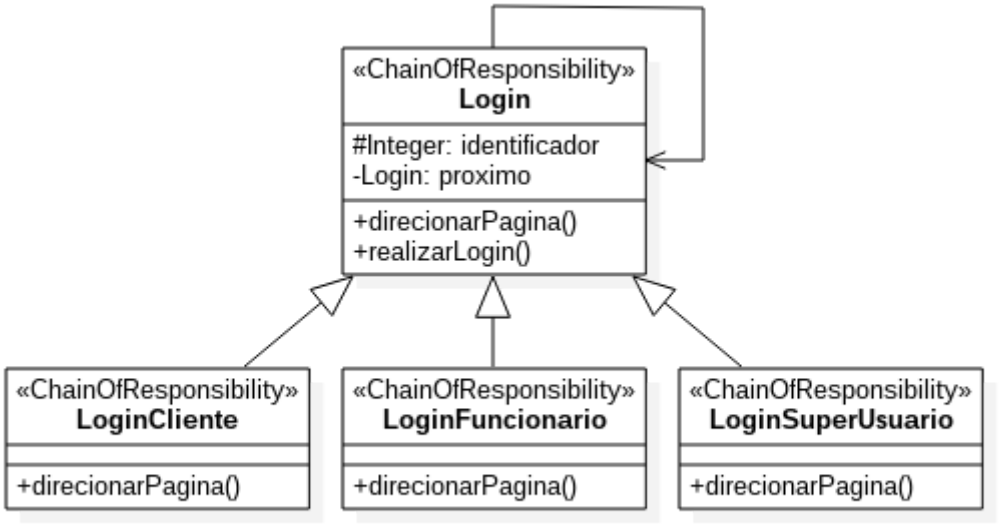
Cássio Henrique Resende Reis - 201576041
Gabriel Martins Santana - 201576002
Mateus Gonçalo do Nascimento - 201576003

- **Chain Of Responsibility**

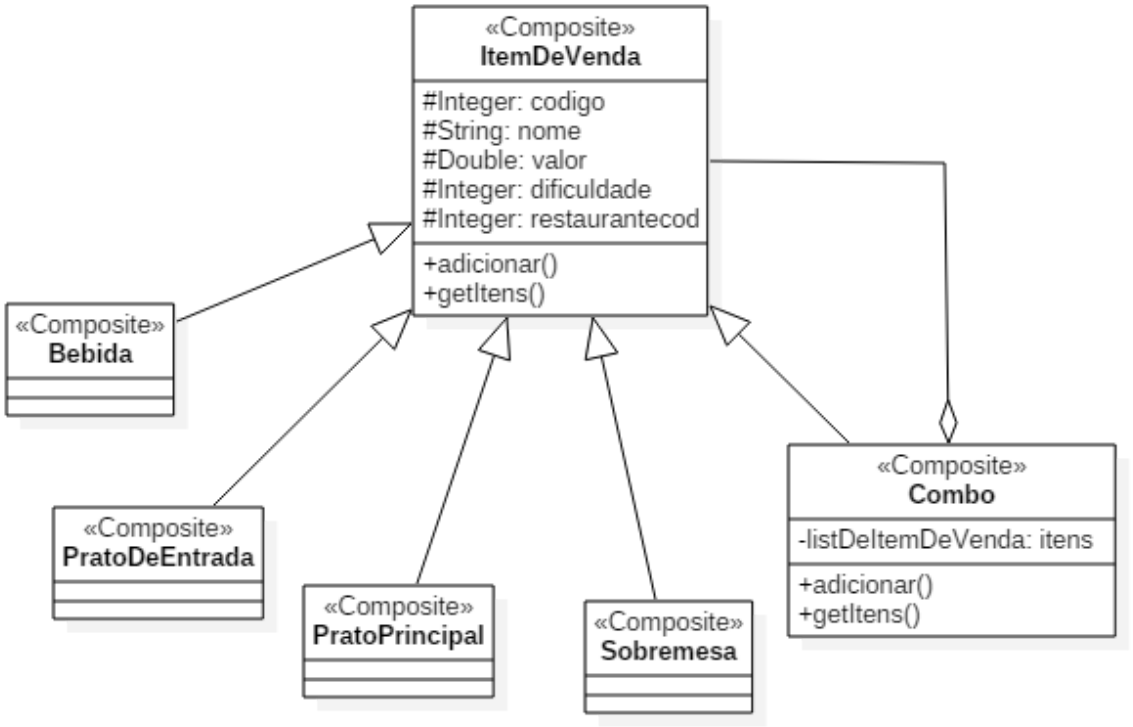


O padrão Chain of Responsibility tem como objetivo evitar a dependência entre o objeto receptor e o objeto solicitante. Este padrão é atendido no nosso trabalho pela realização de login de formas diferentes para cada tipo de usuário do sistema. Dessa forma, cada tipo de usuário só terá acesso a funcionalidades específicas para ele.

No código, temos a classe abstrata Login que dentre seus diversos métodos, implementa o direcionarPagina() que irá verificar o tipo do usuário que fez o login e o redirecionar para a página correspondente a ele.

O uso desse padrão faz com que a estrutura da cadeia de responsabilidades não tenha conhecimento das classes que a compõem, de modo que há um acoplamento fraco.

- **Composite**

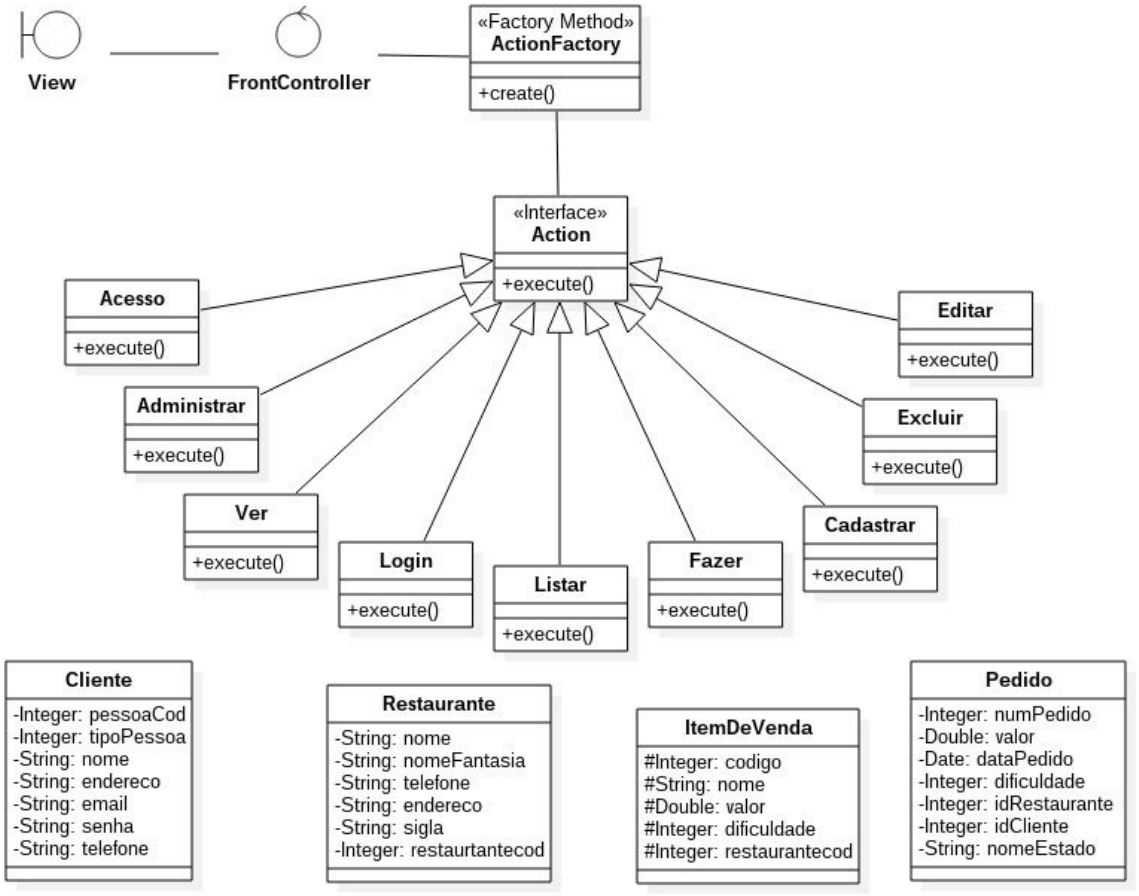


O padrão Composite é utilizado para representar um objeto composto pela união de objetos semelhantes. Este padrão é atendido no cálculo dos preços de combos em nosso sistema. Assim, o agrupamento de diferentes produtos em um combo irá resultar em diferentes preços.

Temos então a classe abstrata ItemDeVenda que possui seus atributos e implementa todos os métodos necessários. As classes Bebida, PratoDeEntrada, PratoPrincipal e Sobremesa herdam da classe ItemDeVenda. Já a classe Combo possui uma lista de itens de venda como atributo, de modo que cada uma das classes que herdam de ItemDeVenda podem ser adicionadas ao Combo, sendo tradas da mesma forma.

Portanto, o padrão Composite permitiu que agrupássemos produtos diferentes, mas com características semelhantes em um único item de venda.

- **Factory Method, Command, Façade, MVC**



No nosso trabalho, os padrões Factory Method, Command, Façade e MVC foram implementados em conjunto. O objetivo do Factory Method é definir a interface de criação de um objeto, mas deixando que as subclasses definam qual classe será instanciada. O Command por sua vez tem como objetivo parametrizar um cliente com diferentes situações, através do encapsulamento de uma solicitação. Já o Façade objetiva definir uma interface única para um conjunto de interfaces do sistema. Por fim, o MVC é um padrão arquitetural, que busca separar a representação da informação da interação do usuário com ele.

No código, a classe ActionFactory cumpre com o padrão de Factory Method ao implementar o método create(), no qual a interface de criação dos objetos será criada. Há também a classe FrontController que serve de interface única para as Actions do sistema, cumprindo assim com o padrão Façade.

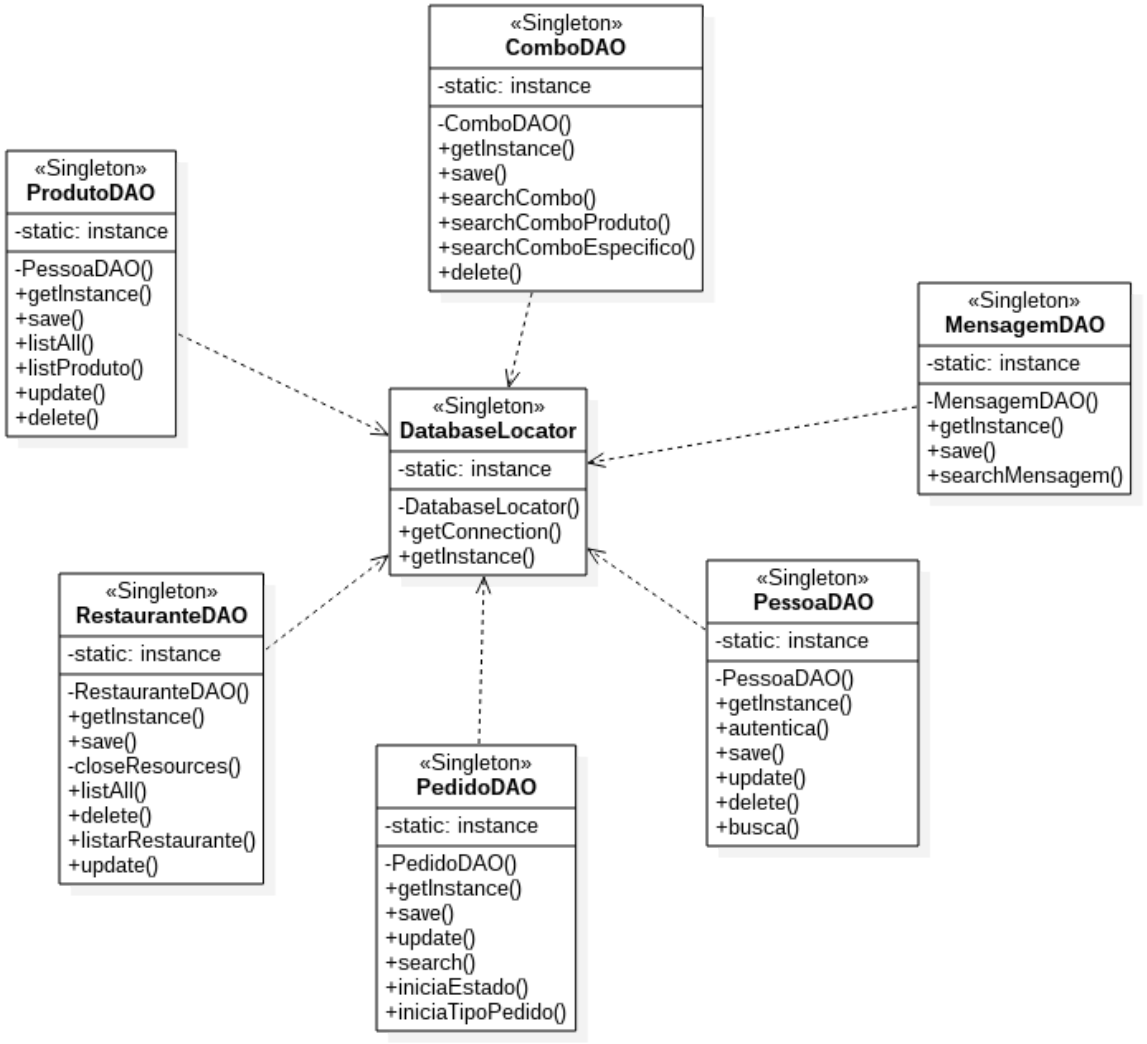
O padrão Command é atendido por todas as classes que implementam a interface Action (que no diagrama estão resumidas, devido à sua grande quantidade) de modo que todas elas possuem uma interface em comum.

Por fim o MVC é atendido pelo fato de haver uma divisão entre a View, o Controller (FrontController) e o Model de nosso sistema.

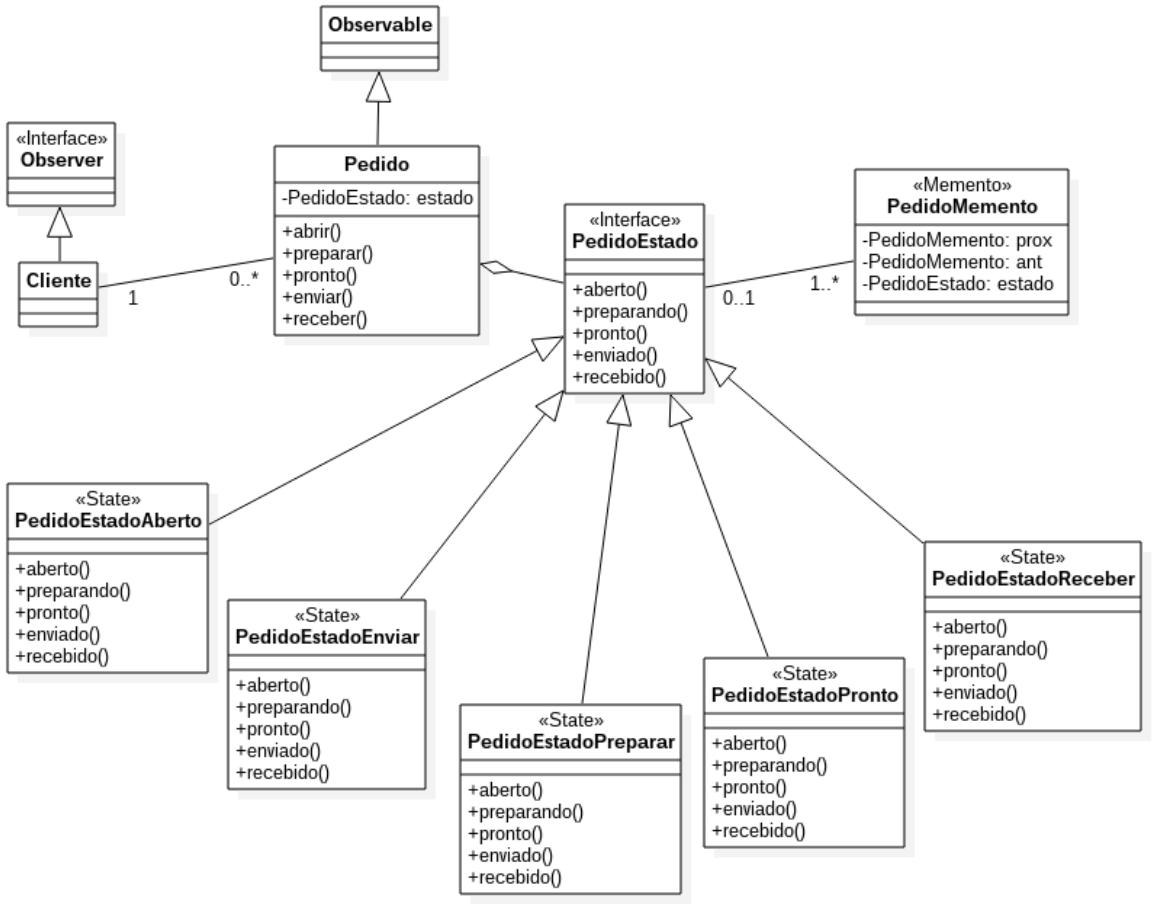
- **Singleton**

O conceito do padrão Singleton foi aplicado em diversas situações no desenvolvimento do trabalho. Nele o objeto basicamente possui uma única instância que é acessada através de um ponto único.

Como o diagrama acima mostra, todas as classes possuem uma instância estática, que pode ser acessada através do método getInstance().



- **State, Observer, Memento**



Os padrões State, Observer e Memento foram implementados em conjunto no nosso sistema. O padrão State tem como objetivo permitir que um objeto mude seu comportamento, conforme mude o seu estado. Já o padrão Observer tem como objetivo definir uma relação de um para muitos entre os objetos, de modo que quando um objeto mudar de estado, todos os seus dependentes serão notificados. Por fim o padrão Memento busca armazenar o estado de um objeto de modo que seja possível restaurá-lo posteriormente.

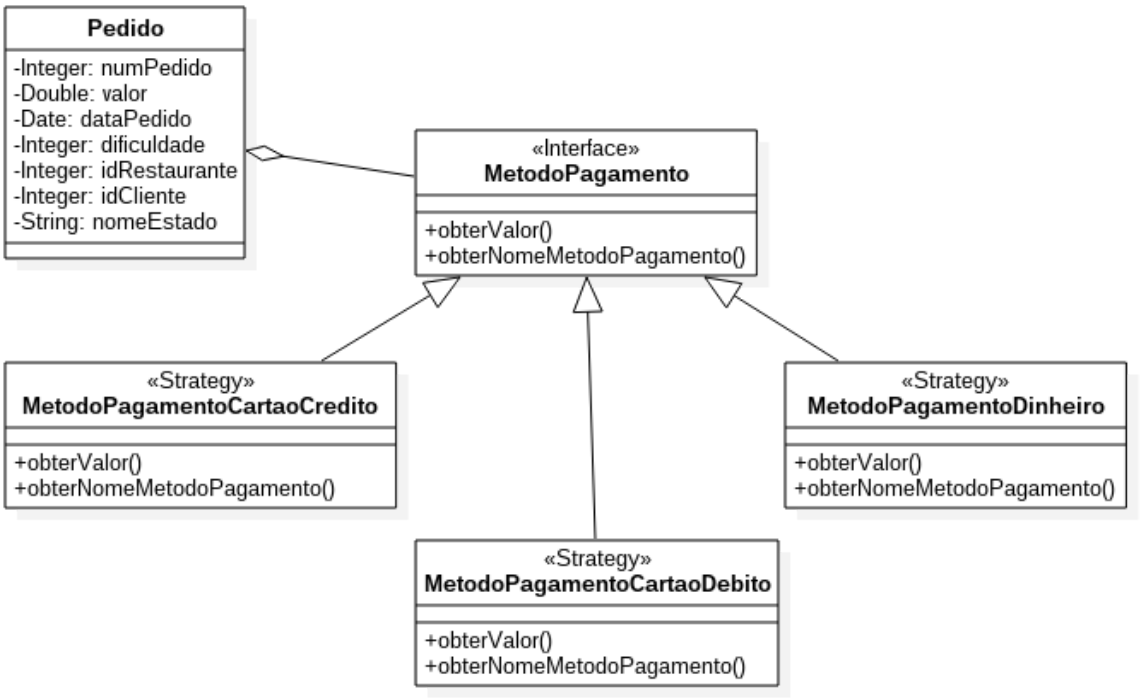
No nosso sistema tais padrões foram implementados na parte de controle dos estados dos pedidos, de modo que o State define comportamentos diferentes do estado de pedido, o Observer notifica os clientes quando ocorre mudança no estado dos pedidos e o Memento permite que os estados possam ser recuperados desde que não tenha ocorrido nenhuma outra recuperação neles antes.

No código, para a utilização do padrão State, temos a interface PedidoEstado que define os métodos a serem implementados nas subclasses. As subclasses são responsáveis por implementar os métodos da interface, contudo apenas o método relativo ao estado a qual ela representa contêm as operações necessárias para mudar o comportamento do objeto.

Já para a utilização do padrão Observer, foi utilizada a interface Observer do java, que é utilizada pelo objeto Cliente. Esta interface tem a responsabilidade de observar o objeto Pedido, que por sua vez é agregado por PedidoEstado, que é o que deseja-se observar.

Por fim o padrão Memento é utilizado através da classe PedidoMemento, que implementa as funcionalidades para que o estado do pedido seja armazenado e recuperado no sistema.

- **Strategy**



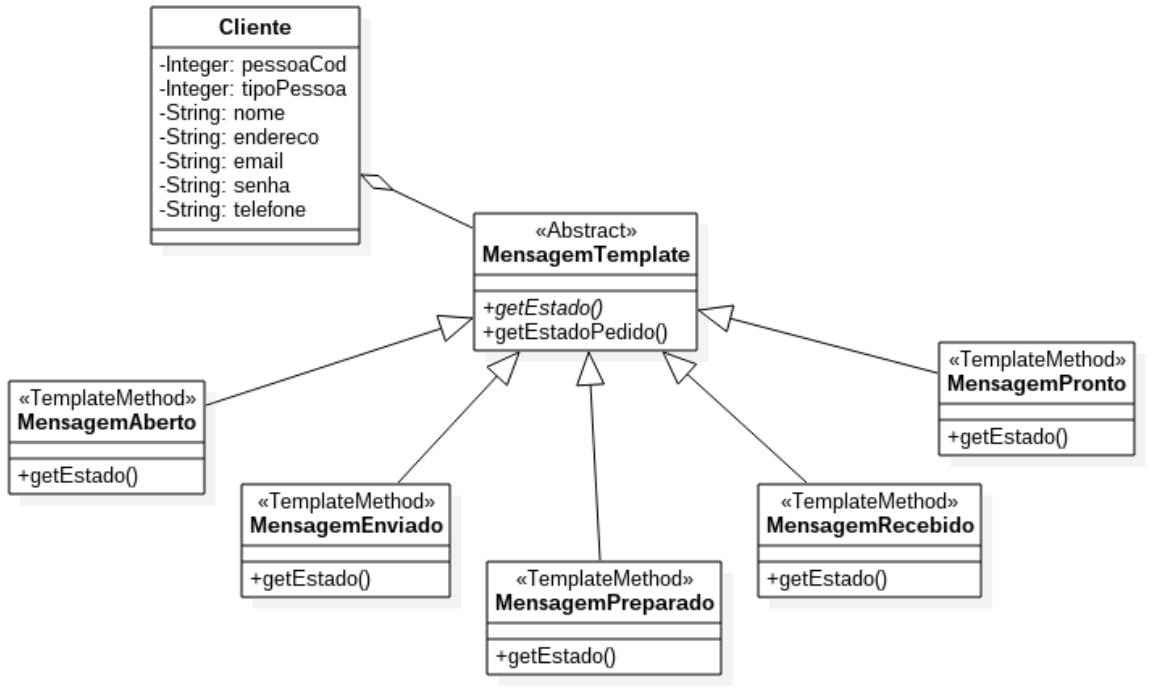
O padrão Strategy é responsável por fazer a distribuição de responsabilidades entre os objetos, de modo que são criadas famílias de objetos encapsuladas e intercambiáveis, ou seja, algoritmos semelhantes são separados do objeto que os utiliza. Este padrão é atendido pela escolha do método de pagamento no nosso sistema.

No código, temos a interface MetodoPagamento que define os métodos obterValor() e obterNomeMetodoPagamento(). As classes que definem os métodos de pagamento com cartão de crédito, cartão de débito e dinheiro implementam os

métodos definidos na interface para cada uma das estratégias de pagamento disponíveis.

Dessa forma, o cálculo do valor de pagamento é realizado em classes diferentes, mesmo que os algoritmos sejam semelhantes, de modo que o objeto que fará uso da estratégia (Pedido) não deverá implementar seus métodos. Assim fica mais fácil fazer a reutilização, extensão e manutenção do código.

- **Template Method**



O Template Method tem como objetivo definir um “molde” de um algoritmo, permitindo que alguns passos sejam realizados nas subclasses. Este padrão é utilizado em nosso trabalho para enviar mensagens ao cliente toda vez que o estado do pedido muda.

No código, temos a classe abstrata MensagemTemplate que implementa os métodos necessários e suas classes herdeiras que são MensagemAberto, MensagemEnviado,MensagemPreparado, MensagemRecebido e MensagemPronto. Assim quando uma mensagem vai ser enviada, o MensagemTemplate é acionado e por sua vez, chama o objeto que possui o “molde” correto para a situação.

Devido ao fato de as operações dos “moldes” serem muito simples, a divisão delas em subclasses torna o código mais fácil de ser modificado e mantido.