JULIO CESAR SILVA LIMA

Template Method

São Paulo

2020

Sumario

1 - Introdução

2 - Funcionamento

3 - Diagrama

4 - Código de Exemplo

5 - Pontos negativos

6 - Pontos negativos

7 - Conclusão

8 - Referências

Introdução

A ideia inicial de padrão de projeto surgiu em 1977 com Christopher Alexander na área de Arquitetura (prédios e cidades). Seus livros e suas ideias foram usados de inspiração para os desenvolvedores de software. Além da arquitetura e do desenvolvimento de software, outras áreas como a Química e as áreas da Engenharia também possuem catálogos de soluções para problemas recorrentes.

Na área do desenvolvimento de software existe um catálogo de soluções desde 1995 com o livro "Design Patterns - Elements of Reusable Object-Oriented Software". Seus idealizadores foram Gamma, Helm, Johnson e Vlissides. Nesse livro são descritos diversos Padrões de Projetos.

Com o catálogo de Padrões de Projetos passamos a ter um vocabulário em comum, soluções descritas e com nomes descritivos, entre diversas outras vantagens.

Padrões de Projetos normalmente possuem Nome, Problema, Solução e Consequências/Forças.

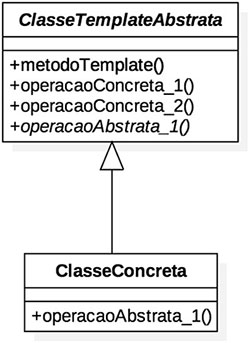
Funcionamento

O Padrão de Projeto Template Method define os passos de um algoritmo e permite que a implementação de um ou mais desses passos seja fornecida por subclasses. Assim, o Template Method protege o algoritmo e fornece métodos abstratos para que as subclasses possam implementá-los.

A definição oficial do padrão Template Method é: “O Padrão Template Method define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo”.

Portanto, o padrão Template Method basicamente oferece um método que define um algoritmo (uma sequência de passos) que pode, por sua vez, ser definido como abstrato para posteriormente ser implementado por uma subclasse. Pode-se notar que a estrutura do algoritmo fica inalterada mesmo com as subclasses fazendo parte da implementação.

Diagrama

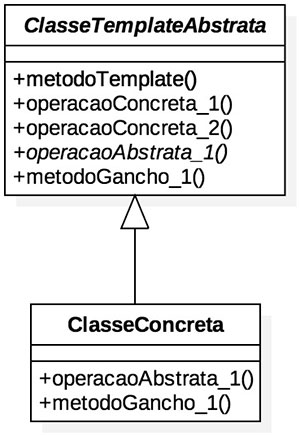


Não está explícito no diagrama, mas o método metodoTemplate() é final, não poderá ser sobrescrito por subclasses de ClasseTemplateAbstrata.

Todos os métodos: operacaoConcreta\_1(), operacaoConcreta\_2() e operacaoAbstrata\_1().

Todos esses são utilizados dentro de metodoTemplate() em uma ordem bem definida e compartilhada por subclasses de ClasseTemplateAbstrata.

Uma variação do diagrama acima que você pode encontrar em outras literaturas de engenharia de software é a seguinte:



Em Template Method métodos gancho ficam na classe de mais alto nível da hierarquia, ClasseTemplateAbstrata.

Eles têm, ou uma implementação vazia (sem linhas de código entre as chaves de abertura e fechamento de bloco) ou uma implementação padrão.

Nesse caso é comum que os métodos: operacaoConcreta\_1() e operacaoConcreta\_2().

Que esses métodos sejam também final como o método metodoTemplate(), ou seja, não podem ser sobrescritos.

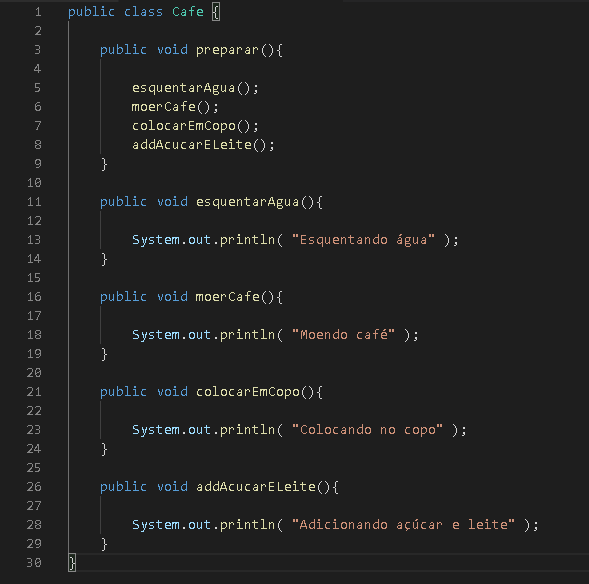
O método metodoGancho\_1() pode ser sobrescrito, mas não é obrigatório. Já o método operacaoAbstrata\_1() têm que ser sobrescrito, pois é abstrato.

Código de Exemplo

Vamos trabalhar com um pequeno projeto de cafeteria, mais precisamente com as classes responsáveis pela criação de café e chá, ambas bebidas cafeinadas (indício de uma superclasse).

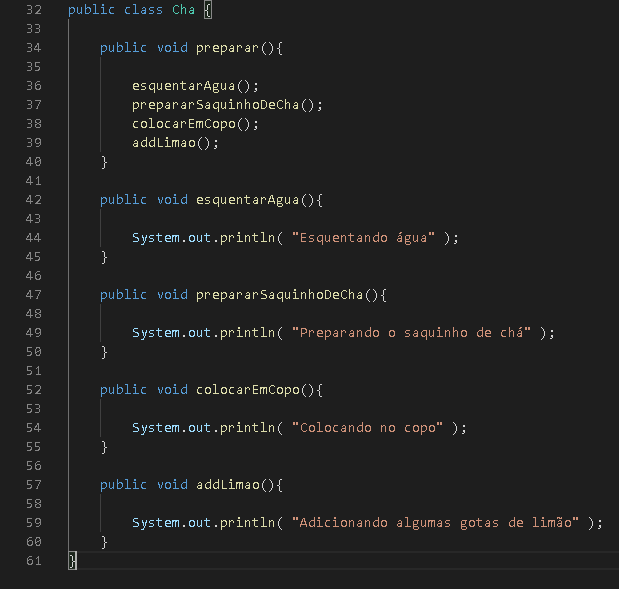
Note que ambos os códigos de classe têm métodos muito similares.

Vamos começar com o código da classe **Cafe**:



Note principalmente os passos (algoritmo) de processamento no método preparar().

Agora a classe **Cha**:



Apesar de os métodos nem sempre terem os mesmos nomes em ambas as classes eles são similares em execução (moer café para um café é similar a preparar o saquinho de chá para um chá).

Como informado, o método *preparar()* é exatamente o mesmo, digo, mudando alguns nomes de métodos, ele tem exatamente os mesmos passos necessários para ambas as classes.

Veja como é evidente o problema de repetição de código em ambas as classes.

Note que muitos dos padrões de projeto que existem hoje em dia têm, em sua maioria, um único objetivo:

Reduzir ou acabar de vez com os códigos duplicados.

Código duplicado é equivalente a:

* Manutenção mais lenta e crítica. Corrigindo ou acrescentando algo você terá de lembrar de atualizar também o trecho de código igual em outras partes do projeto;
* Maior consumo de memória no processamento. Mais códigos (mesmo que repetidos), mais espaços e quantidade de recursos como necessidade de execução;
* Aumento dos custos monetários de evolução do projeto, pois a leitura e atualização do código ficam mais lentos.

Provavelmente tem "n" outros problemas, mas os indicados acima já nos ajudam a entender o porquê de tantos padrões para remover códigos repetidos.

Com isso podemos seguir a solução: aplicar o padrão Template Method.

Sabemos que as classes são relacionadas, ambas são classes de bebidas cafeínadas, logo podemos aproveitar esse ponto e entender melhor os nomes dos métodos com execuções equivalentes, porém com rótulos distintos.

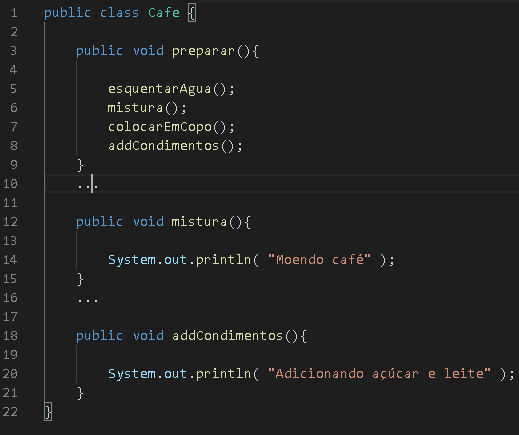
Os métodos moerCafe() e addAcucarELeite() da classe Cafe e os métodos prepararSaquinhoDeCha() e addLimao() da classe Cha são compatíveis em suas assinaturas para que a criação de um template method seja possível.

Vamos renomear os métodos moerCafe() e prepararSaquinhoDeCha() para mistura().

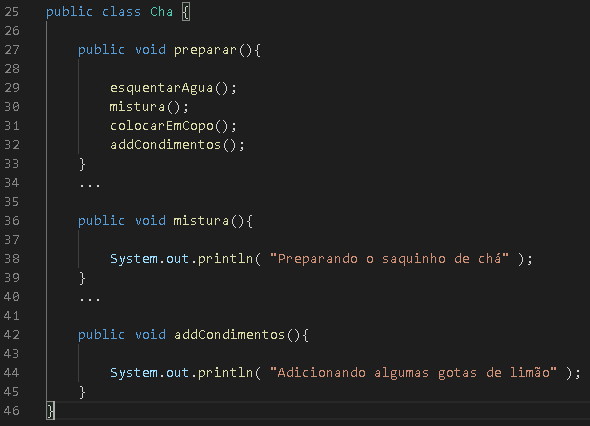
Para os métodos addAcucarELeite() e prepararSaquinhoDeCha() temos um nome mais comum a ação deles: addCondimentos().

Com isso segue as novas configurações de classe.

Primeiro a classe **Cafe**:



Classe **Chá**:

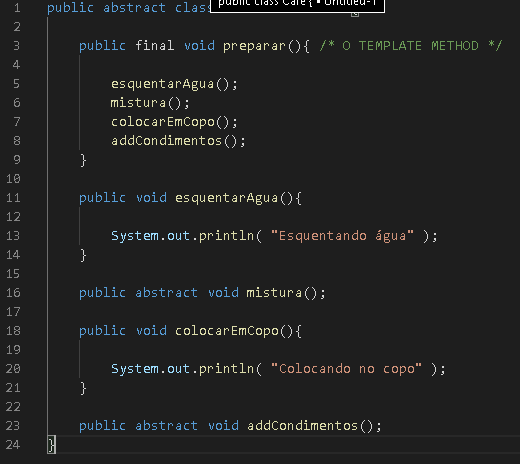


Com isso podemos prosseguir criando uma superclasse para as bebidas.

Antes, note a importância dos passos executados em preparar(). A ordem é fundamental e igual em ambas as classes.

Esse será nosso Template method ou, como também conhecido em algumas literaturas de engenharia de software: método gabarito.

Podemos assim criar a superclasse **BebidaCafeinada**:

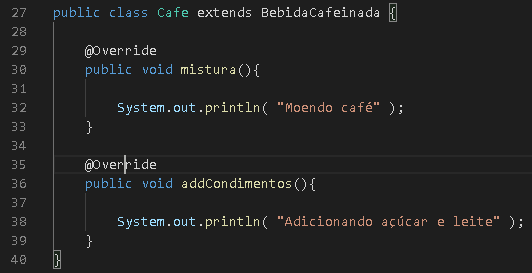


Como informado no diagrama do padrão, nosso template Method é final, não pode ser sobrescrito.

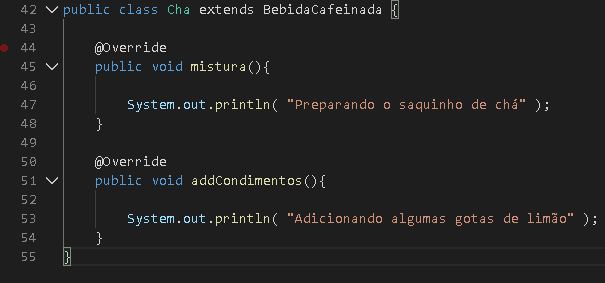
Quanto aos métodos concretos, se tiver o conhecimento de que eles sempre deverão apresentar processamentos iguais, os coloque também como final.

Como em mistura() e em addCondimentos() temos processamentos diferentes nas subclasses, os colocamos como abstract. Assim podemos ir às novas configurações das classes Cafe e Cha.

Começando com **Cafe**:

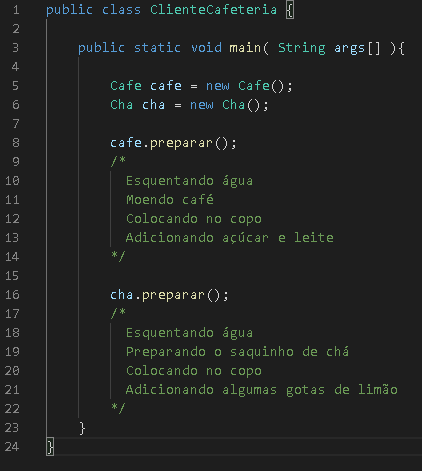


E **Cha**:



Assim removemos códigos duplicados, conseguindo a atualização, quando necessária, em apenas um ponto.

Vamos a um código cliente de exemplo:



Destrinchando a chamada a preparar() da variável cafe temos:

1. Invoca preparar() que está definido em BebidaCafeinada;
2. Invoca esquentarAgua() que está definido em BebidaCafeinada;
3. Invoca mistura() que está definido na classe especializada, Cafe;
4. Invoca colocarEmCopo() que está definido em BebidaCafeinada;
5. Invoca addCondimentos() que está definido na classe especializada, Cafe.

E assim prossegue nos mesmos moldes para a instância de Cha.

É muito importante que você saiba a partir desse ponto o que é um Template Method, ele apenas define a ordem exata das chamadas de métodos no algoritmo

Ponto negativo

* Se muitos métodos da classe que contém o método template forem abstratos, isso pode atrapalhar o projeto em termos de consumo de memória.

Pontos positivos

* Padrão de projeto quando aplicado, cria uma linguagem universal para os programadores, aumentando assim a eficiência na leitura do código;
* Remove substancialmente o número de linhas de código duplicadas.

Conclusão

Esse padrão é daqueles que pouco têm itens contra sua implementação, até porque o principal objetivo dele é remover código duplicado.

Lembrando que esse padrão vai permitir definir os passos exatos de um algoritmo (método) onde alguns desses passos, mais específicos, são implementados por subclasses.

Referencias

* <https://www.thiengo.com.br/padrao-de-projeto-template-method-metodo-template>
* <https://www.devmedia.com.br/padrao-de-projeto-template-method-em-java/>