

Semana 07

Composite Design Pattern

WebQuest

Introdução

Template Method is a **behavioral design pattern** and it's used to create a method stub and deferring some of the steps of implementation to the subclasses. **Template method** defines the steps to execute an algorithm and it can provide default implementation that might be common for all or some of the subclasses. It provides a template or a structure of an algorithm which is used by users. A user provides its own implementation without changing the algorithm's structure.



Let's understand this pattern with an example, suppose we want to provide an algorithm to build a house. The steps need to be performed to build a house are - building foundation, building pillars, building walls and windows. The important point is that we can't change the order of execution because we can't build windows before building the foundation. So in this case we can create a template method that will use different methods to build the house.

Tarefa

Conhecer e experimentar o Template Method Pattern, implementando um exemplo.

Processo: Fazer pelo menos até o item 3 em sala de aula [35 min]

1. [Em dupla] Implementar e testar a classe `WoodenHouse` com uma classe de teste `HousingClient`. O método `buildHouse()` pode ficar sem o "final"? O que o "final" acarreta no código?!?

R: Primeiramente, o complemento final significa que o método implementado não pode ser sobrescrito por outras classes filhas. Dessa forma, o complemento final serve para prevenir qualquer má extensão nesse caso.

```
public class WoodenHouse {
    public final void buildHouse(){
        buildFoundation();
        buildPillars();
        buildWalls();
        buildWindows();
        System.out.println("House is built.");
    }
    private void buildFoundation() {
        System.out.println("Building foundation with cement,iron rods
and sand");
    }
    private void buildPillars() {
        System.out.println("Building Pillars with Wood coating");
    }
    private void buildWalls() {
        System.out.println("Building Wooden Walls");
    }
    private void buildWindows() {
        System.out.println("Building Glass Windows");
    }
}

public class HousingClient {
    public static void main(String[] args) {
        WoodenHouse woodenhouse = new WoodenHouse();
        woodenhouse.buildHouse();
        System.out.println("*****");
    }
}
```

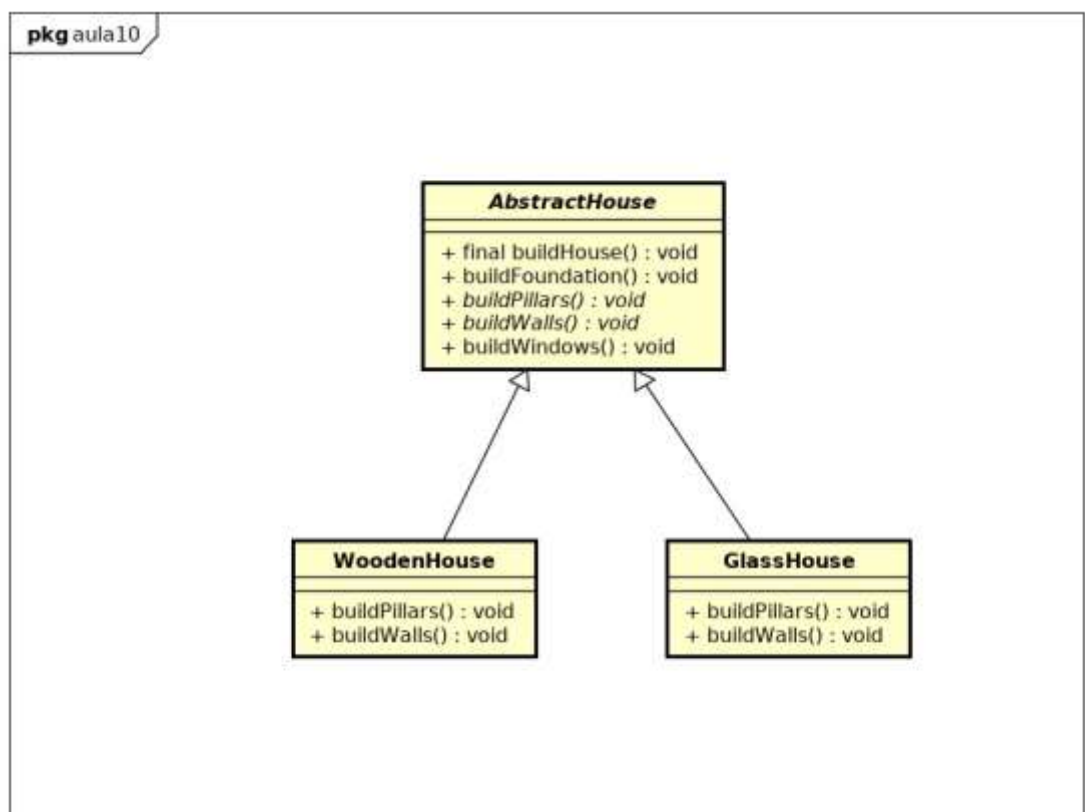
2. Inclua uma nova classe `GlassHouse`, idêntica a `WoodenHouse`, exceto pelos métodos `buildPillars()` e `buildWalls()`, em que se troca "Wood" por "Glass". Implemente e teste a construção de uma `WoodenHouse` e de uma `GlassHouse`, refatorando a classe teste `HousingClient`.
3. Claramente há um mau cheiro: os dois códigos das duas classes são redundantes. Faça uma refatoração que use o padrão Template

Method e elimine o mau cheiro, em que se considera que apenas os métodos `buildPillars()` e `buildWalls()` é que serão sobrepostos (overridden) nas subclasses do padrão. Faça o seguinte:

- a. Implemente e teste o sistema na nova estrutura. Você deve manter o método `buildHouse()` como final nessa reestruturação? A classe de testes `HousingClient` precisará ser refatorada também?!

R: O complemento final foi mantido na classe abstrata `AbstractHouse` para justamente prevenir que uma extensão do programa rompa com o Template Method. Não, pois o Template Method simplesmente otimiza o código evitando repetições e mudanças de comportamento.

- b. Apresente o novo diagrama de classes de acordo com a nova estrutura arquitetural. Ele pode ser feito à mão, fotografado e inserido no texto como figura.



powered by Astah

- c. Neste caso, identifique o método template, a classe template, os métodos hook, a classe hook e os métodos que não são nada disso.

R: Baseado no diagrama UML acima, pode-se notar que existe um Template Method envolta do método `buildHouse()` que possui implementação de seu algoritmo na classe abstrata, porém alguns dos seus sub-métodos são abstratos. Com isso, a nossa classe Template é a classe `AbstractHouse`, enquanto que as classes hook serão `WoodenHouse` e `GlassHouse`. Por fim, os métodos hook que ligam

essas classes são os métodos buildPillar() e buildWalls(), os quais são implementados nas classes concretas. Ademais, os métodos buildFoundation() e buildWindows() mantiveram implementados na classe abstrata, pois ambas suas filhas utilizariam a mesma implementação.

4. Coloque as subclasses do item 3 numa hook class. Faça o seguinte:
 - a. Implemente e teste o sistema na nova estrutura. A classe de testes HousingClient precisará ser refatorada também?!
 - b. Apresente o novo diagrama de classes de acordo com a nova estrutura arquitetural. Ele pode ser feito à mão, fotografado e inserido no texto como figura.
 - c. Neste caso, identifique o método template, a classe template, os métodos hook, a classe hook e os métodos que não são nada disso.
5. Combine o padrão Builder com o Template Method usado neste WebQuest. Faça o seguinte:
 - a. Implemente e teste o sistema na nova estrutura. A classe de testes HousingClient precisará ser refatorada também?!
 - b. Apresente o novo diagrama de classes de acordo com a nova estrutura arquitetural. Ele pode ser feito à mão, fotografado e inserido no texto como figura.
 - c. Neste caso, identifique o método template, a classe template, os métodos hook, a classe hook, os métodos que não são nada disso, a classe produto do builder, a classe abstrata do builder e as classe concretas do builder.
6. Responda ao seguinte:
 - a. Por que usualmente o método buildHouse() tem que ser "final"?
 - b. Os métodos template e hook podem ser abstratos e concretos?!
 - c. Um método template pode ser ao mesmo tempo um método hook?
7. Gerar documento com a resposta dos item 1 a 6 e postá-lo, com os nomes dos membros do dupla, na atividade correspondente do TIDIA num dos membros do dupla.

Recursos

1. https://sourcemaking.com/design_patterns/template_method