Le pattern Template Method (Patron de méthode)

Membres
Mady CAMARA
Alseny DIALLO
Assane DIATTA

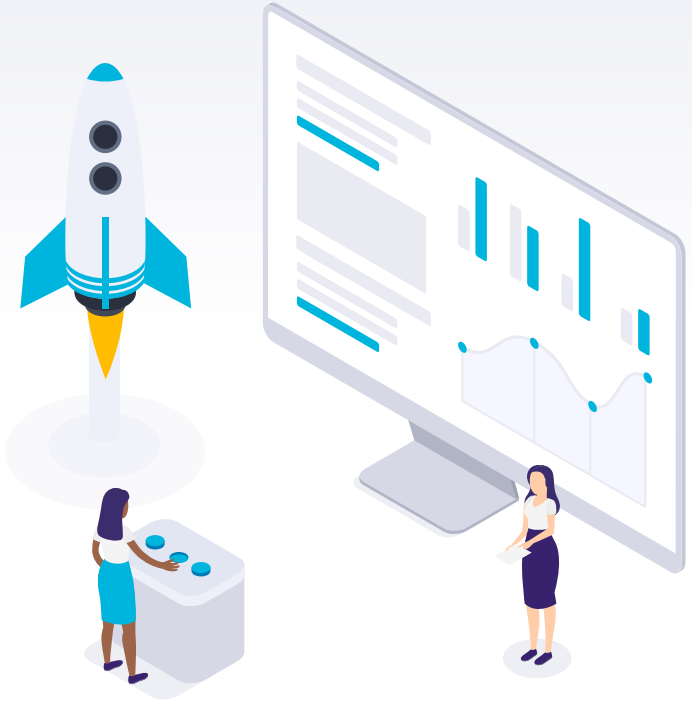
Plan

- 1. Présentation
- 2. Problème
- 3. Solution
- 4. Structure
- 5. Implémentation
- 6. Avantages et inconvénients
- 7. Différences entre strategy pattern

1

Présentation

Patron de méthode



Qu’est-ce que patron méthode?

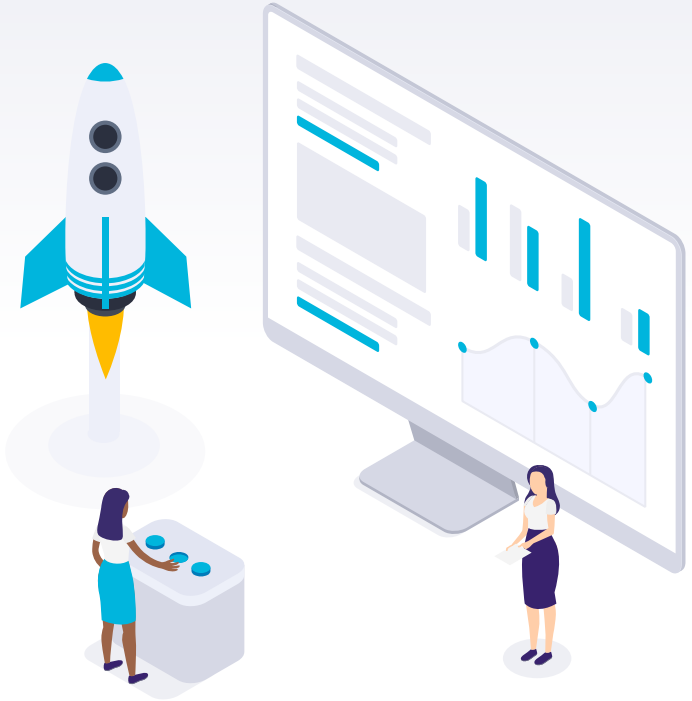
- Patron de méthode est un pattern de comportement.
- Il définit un algorithme comme squelette d’opérations et laisse les détails à implémenter par les classes filles.
- La structure globale et la séquence de l’algorithme sont préservées par la classe parent.
- Populairement utilisé dans le développement du Framework.
- Permet également d’éviter la duplication de code, l’implémentation commune et les étapes sont dans la classe de base.

Discussion

D’une manière générale, il permet de factoriser les comportements d’un algorithme communs a différentes classes d’une hiérarchie, ces dernières se contentant d’implémenter les parties variables.
On distingue généralement deux types de méthodes :

- invariantes : qui doivent absolument être redéfinies par les classes concrètes; on les nomme parfois méthodes de rappel ou hook en anglais
- variantes : qui disposent d’une définition par défaut

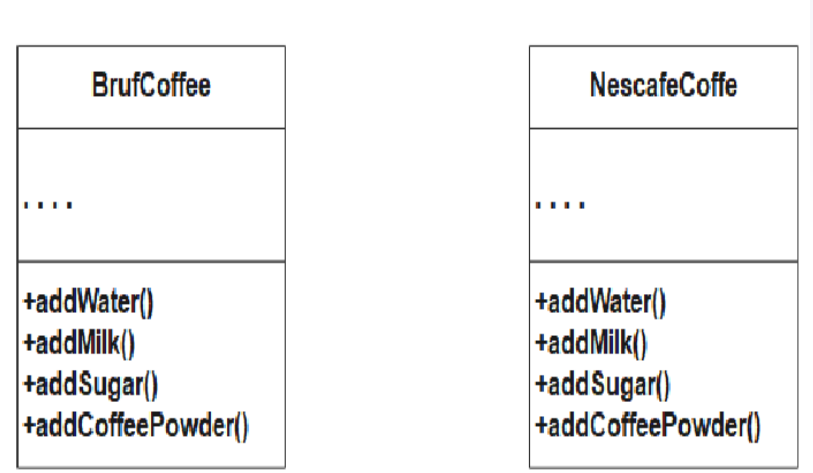
2 Problème



Problème

Supposons que vous voudrez programmer une machine qui prépare du café (disons BruCoffee). Ensuite, vous devez suivre certaines étapes ou procédures telles que faire bouillir de l'eau, ajouter du lait, ajouter du sucre et ajouter du café brut. Plus tard vous voudrez que votre programme préparer du café Nescafé. Ensuite, vous devez également suivre la même procédure que BruCofee, comme faire bouillir de l'eau, ajouter du lait, ajouter du sucre et ajouter du café Nescafé.

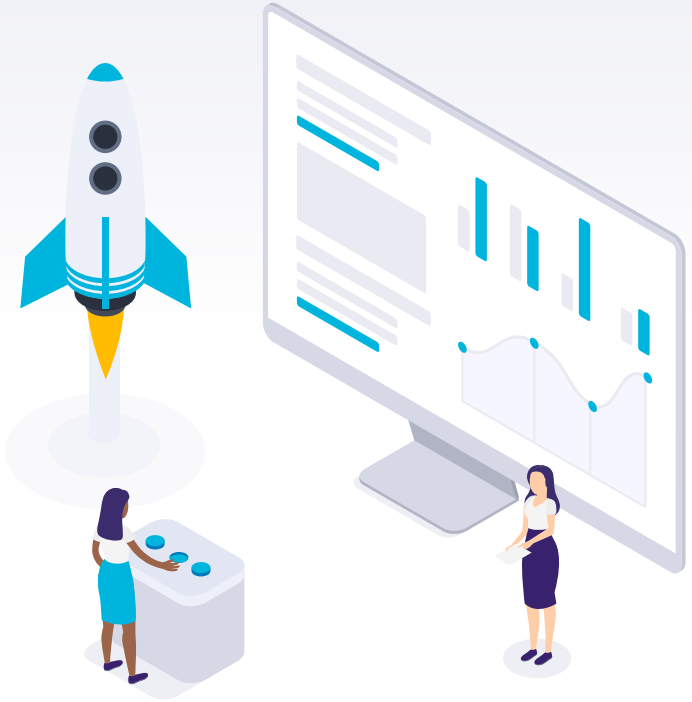
Problème



Au bout d'un moment, vous remarquez que ces deux classes comportent des codes similaires. Bien que le code qui gère les différents types de café soit complètement différent d'une classe à l'autre. Ne serait-ce pas super de se débarrasser de tout ce code dupliqué tout en laissant la structure de l'algorithme intact ?

3

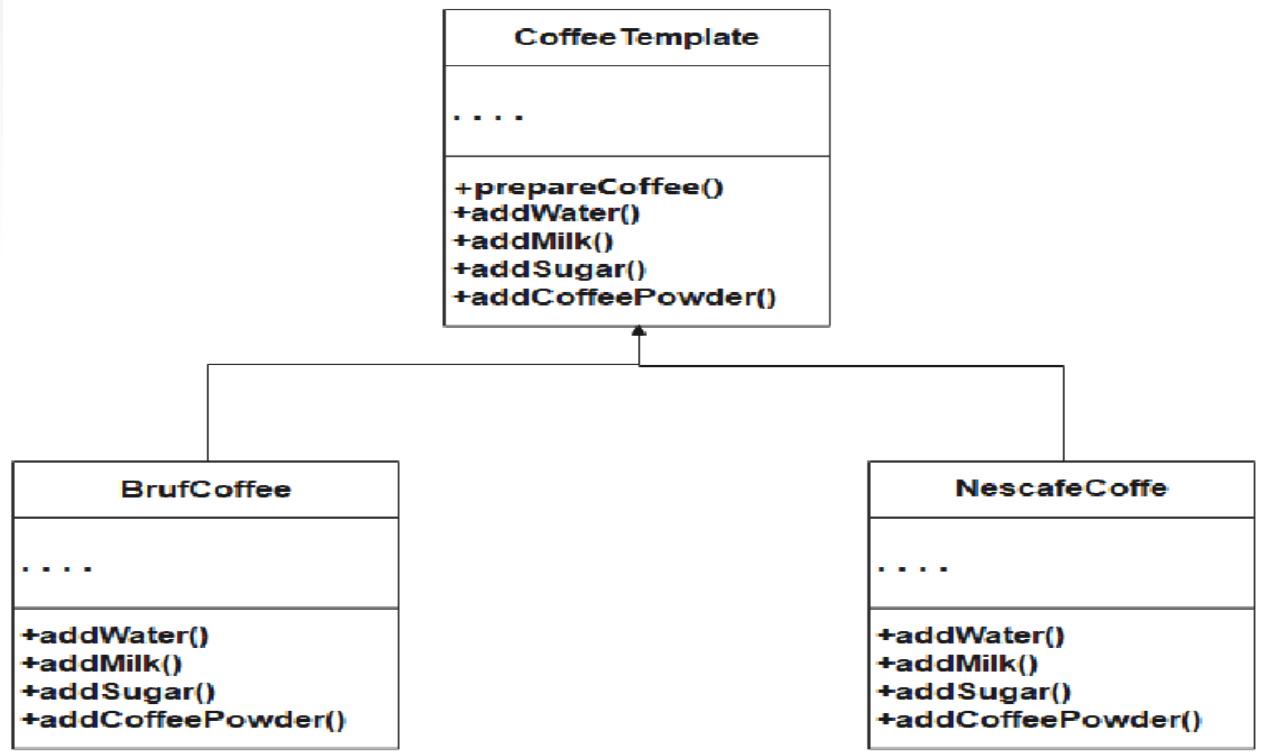
Solution



Solution

Le patron de méthode vous propose de découper un algorithme en une série d'étapes, de transformer ces étapes en méthodes et de mettre l'ensemble des appels à ces méthodes dans une seule méthode socle, le patron de méthode. Les étapes peuvent être abstraites ou avoir une implémentation par défaut. Pour utiliser l'algorithme, le client doit fournir sa propre sous-classe, implémenter toutes les étapes abstraites et redéfinir certaines d'entre elles si besoin (mais pas la méthode socle elle-même).

Solution


CoffeeTemplate
. . . .
+prepareCoffee()
+addWater()
+addMilk()
+addSugar()
+addCoffeePowder()
BrufCoffee
. . . .
+addWater()
+addMilk()
+addSugar()
+addCoffeePowder()
NescafeCoffe
. . . .
+addWater()
+addMilk()
+addSugar()
+addCoffeePowder()

4

Structure

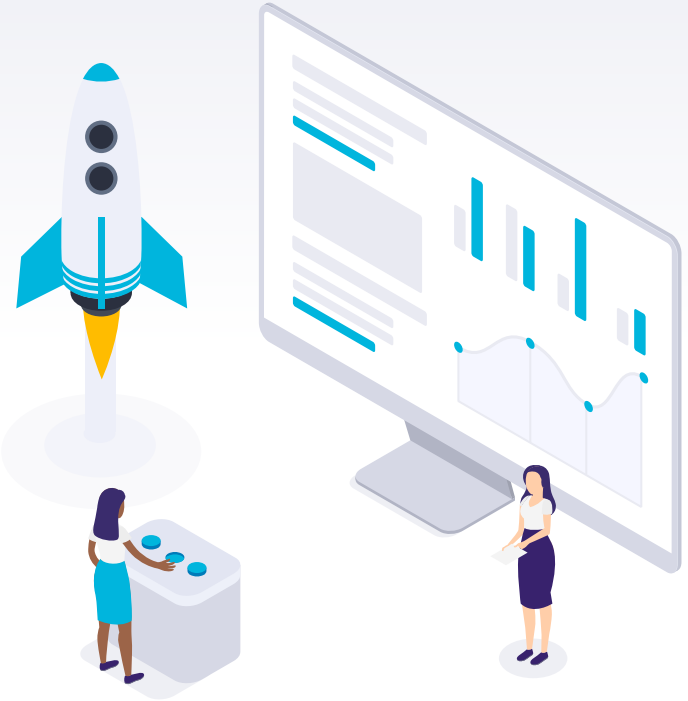
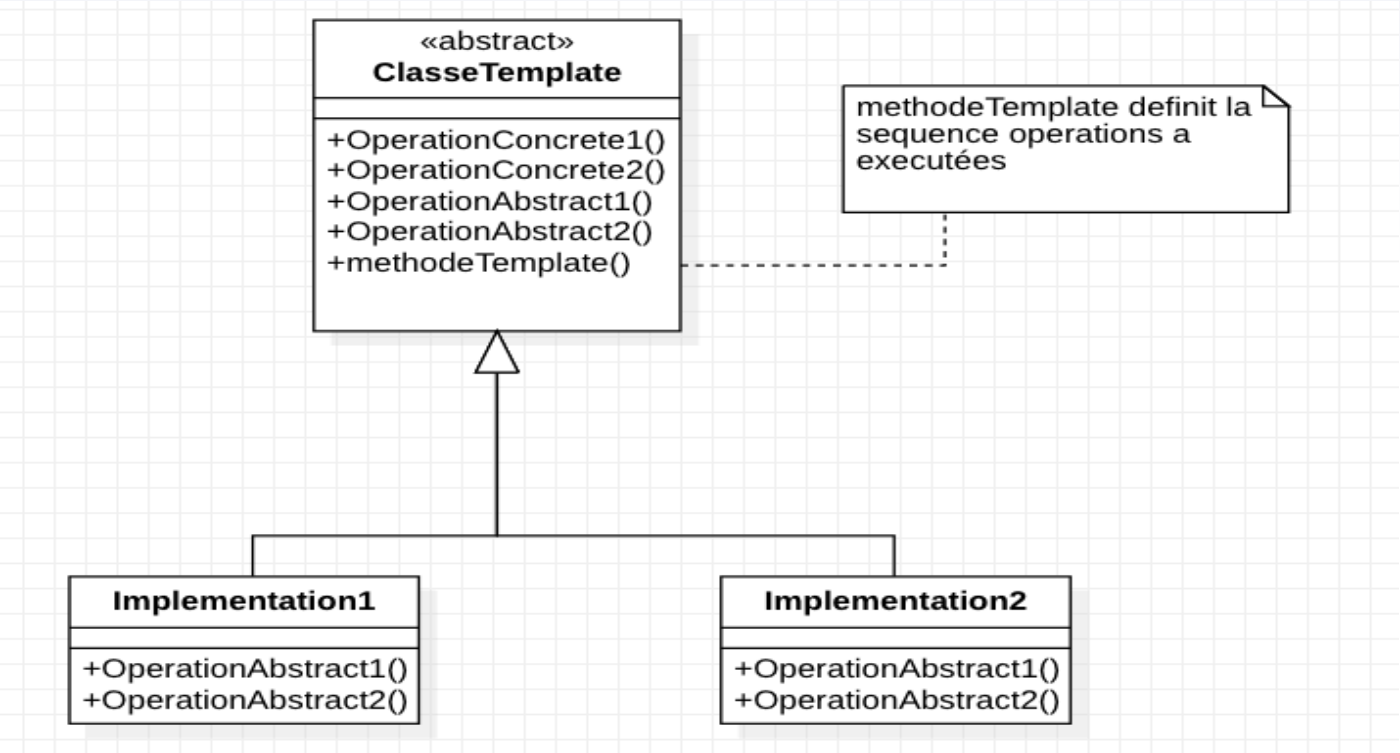


Diagramme de classe



Explication

La Classe Abstraite (**classeTemplate**) déclare des méthodes (qui représentent les étapes d'un algorithme) et la méthode patron de Méthode qui appelle toutes ces méthodes dans un ordre spécifique. Les étapes peuvent être déclarées abstraites ou posséder une implémentation par défaut.

Les Classes Concrètes (**Implementation1**, **Implementation2**) peuvent redéfinir toutes les étapes, sauf la methode (**methodeTemplate**)

5 Implémentation

- **1 Machine a café en grain : classe de base**

ClasseTemplate.java

```
1  abstract class ClasseTemplate{
2
3      public final void templateMethode() {
4          mettreEau();
5          mettreCafe();
6          preparerCafe();
7          servirCafe();
8      }
9
10     protected abstract void mettreEau();
11
12     protected abstract void mettreCafe();
13
14     private void preparerCafe(){
15        System.out.println("Chaufer l'eau");
16        System.out.println("Mettre le café dans l'eau");
17        System.out.println("Melanger le café et l'eau");
18     }
19
20     private void servirCafe(){
21         System.out.println("Mettre le café dans la tasse");
22     }
23
24 }
```

- **1 Machine a café en grain : Les classes concrètes**

Implementation1.java

```java
class Implementation1 extends ClasseTemplate{
    public void mettreEau(){
        System.out.println("Mettre 0,5 litre d'eau");
    }

    public void mettreCafe(){
        System.out.println("Mettre un peu de café touba");
    }

}
```

Implementation2.java

```java
class Implementation2 extends ClasseTemplate{
    public void mettreEau(){
        System.out.println("Mettre 0,7 litre d'eau");
    }

    public void mettreCafe(){
        System.out.println("Mettre du café italien");
    }

}
```

- **1 Machine a café en grain : la classe test**

TestTemplateMethode.java

```
1 public class TestTemplateMethode{
2     public static void main(String args[]){
3         ClasseTemplate impl1 = new Implementation1();
4         ClasseTemplate impl2 = new Implementation2();
5         impl1.templateMethode();
6         impl2.templateMethode();
7     }
8 }
```

- **1 User template : classe de base**

```
1  abstract class UserTemplate {
2
3      String userName;
4      String password;
5
6      UserTemplate() {}
7
8      public void start() {
9          if(logIn(userName, password))
10         {
11             homePage();
12             logOut();
13         }
14
15     }
16
17     abstract boolean logIn(String userName, String password);
18
19     abstract void logOut();
20
21     abstract void homePage();
22 }
```

- **1 User template : Les classes concrètes**

```java
public class Admin extends UserTemplate {
    Admin(String userName, String password){
        this.userName = userName;
        this.password = password;
    }

    public boolean logIn(String userName, String password) {
        System.out.println("Administrateur bien connecté!");
        return true;
    }

    public void logOut() {
        System.out.println("Administrateur bien déconnecté!");
    }

    public void homePage() {
        System.out.println("Bienvenue sur votre DashBoard cher Administrateur !");

    }
}
```

```java
public class Guest extends UserTemplate {
    Guest(String loginCode){
        this.userName = "Invité" ;
        this.password = loginCode;
    }

    public boolean logIn(String userName, String password) {
        System.out.println("Bienvenue sur le compte Invité");
        return true;
    }

    public void logOut() {
        System.out.println("Suppression de toutes les données !");
        System.out.println("Déconnexion Réussie !");
    }

    public void homePage() {
        System.out.println("Bienvenue sur le compte test Invité de notre application");
        System.out.println("NB: En vous déconnectant toutes vos données seront supprimées.");
    }
}
```

- # 1 User template : classe test

```java
import java.util.Scanner;
public class Demo {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        UserTemplate user = null;
        String userName;
        String password;

        System.out.println("\nChoisissez votre type de compte.\n" +
                "1 - Administrateur\n" +
                "2 - Utilisateur\n" +
                "3 - Invité");
        System.out.println("---------------");
        int choice = Integer.parseInt(scan.nextLine());
        switch (choice) {
            case 1:
                System.out.print("Donnez votre Identifiant : ");
                userName = scan.nextLine();
                System.out.print("Donnez votre mot de passe : ");
                password = scan.nextLine();
                user = new Admin(userName, password);
                user.start();
                break;
            case 2:
                System.out.print("Donnez votre Identifiant : ");
                userName = scan.nextLine();
                System.out.print("Donnez votre mot de passe : ");
                password = scan.nextLine();
                user = new User(userName, password);
                user.start();
                break;
            case 3:
                System.out.print("Donnez votre code de connexion Invité : ");
                password = scan.nextLine();
                user = new Guest(password);
                user.start();
                break;
            default:
                System.out.println("Choix Indisponible.");
                break;
        }
    }
}
```

6

Différences entre strategy pattern

Avantages et inconvénients

- Il n'y a pas de duplication de code.
- La réutilisation du code se produit avec le template method car il utilise l'héritage et non la composition. Seules quelques méthodes doivent être remplacées.
- La flexibilité permet aux sous-classes de décider comment implémenter les étapes d'un algorithme.

- Le débogage et la compréhension de la séquence de flux dans le pattern peuvent parfois être déroutants. Vous pouvez finir par implémenter une méthode qui ne devrait pas être implémentée ou ne pas implémenter du tout une méthode abstraite
- Sa Personnalisation est limitée
- Difficile à déboguer et à maintenir

7

Avantages et inconvénients

Le patron de méthode est utilisé lorsqu'une opération particulière a un ou plusieurs comportements invariants qui peuvent être définis en termes d'autres comportements primitifs variables.

La classe abstraite définit le ou les comportements invariants, tandis que les classes d'implémentation définissent les méthodes dépendantes.

“

Indépendantes, chaque classe d’implémentation définit le comportement et il n’y a pas de code partagé entre elles. Les deux sont des modèles de comportement et, en tant que tels, sont consommés à peu près de la même manière par les clients. Le pattern comporte généralement une seule méthode publique la méthode execute() Les Templates peuvent définir un ensemble de méthodes publiques ainsi qu’un ensemble de primitives privées de support que les sous-classes doivent implémenter.

MERCI! 😉

Nos github :

- @Mady-Camara
- @alseny-diallo
- @A-San96