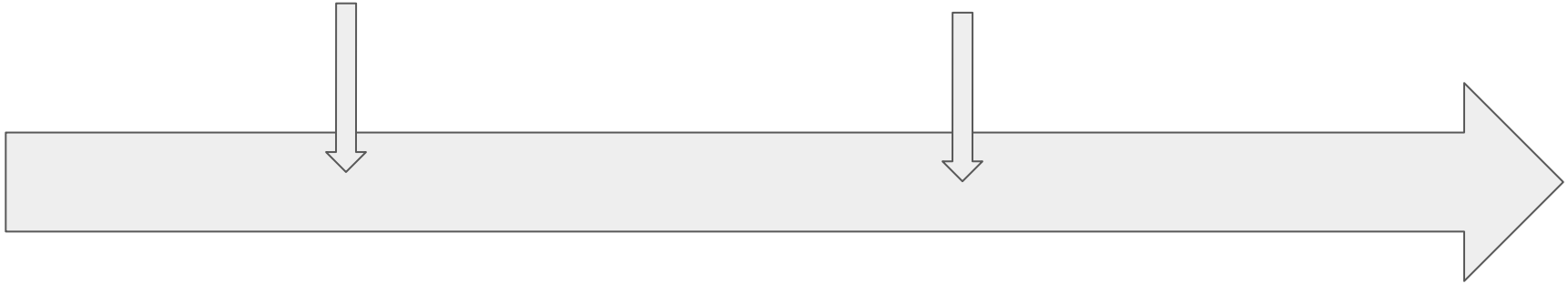
Design pattern builder

Sommaire

Histoire des design patterns

1970 : Christopher Alexander écrit “A pattern Language”

1995 : GoF écrit “Design Patterns - Elements of reusable Object Oriented Software”



Les designs patterns ou patrons de conception

Qu'est-ce qu'un pattern ?

Un pattern désigne un modèle, une structure, etc….

C'est un phénomène ou une organisation qui se répète lors de l'étude de certains sujets.

- Types de patterns
    - Architecture
    - Idiotisme
    - Conception
        - Créateurs
            - Builder pattern
        - Structuraux
        - Comportementaux

Structure d’un design pattern

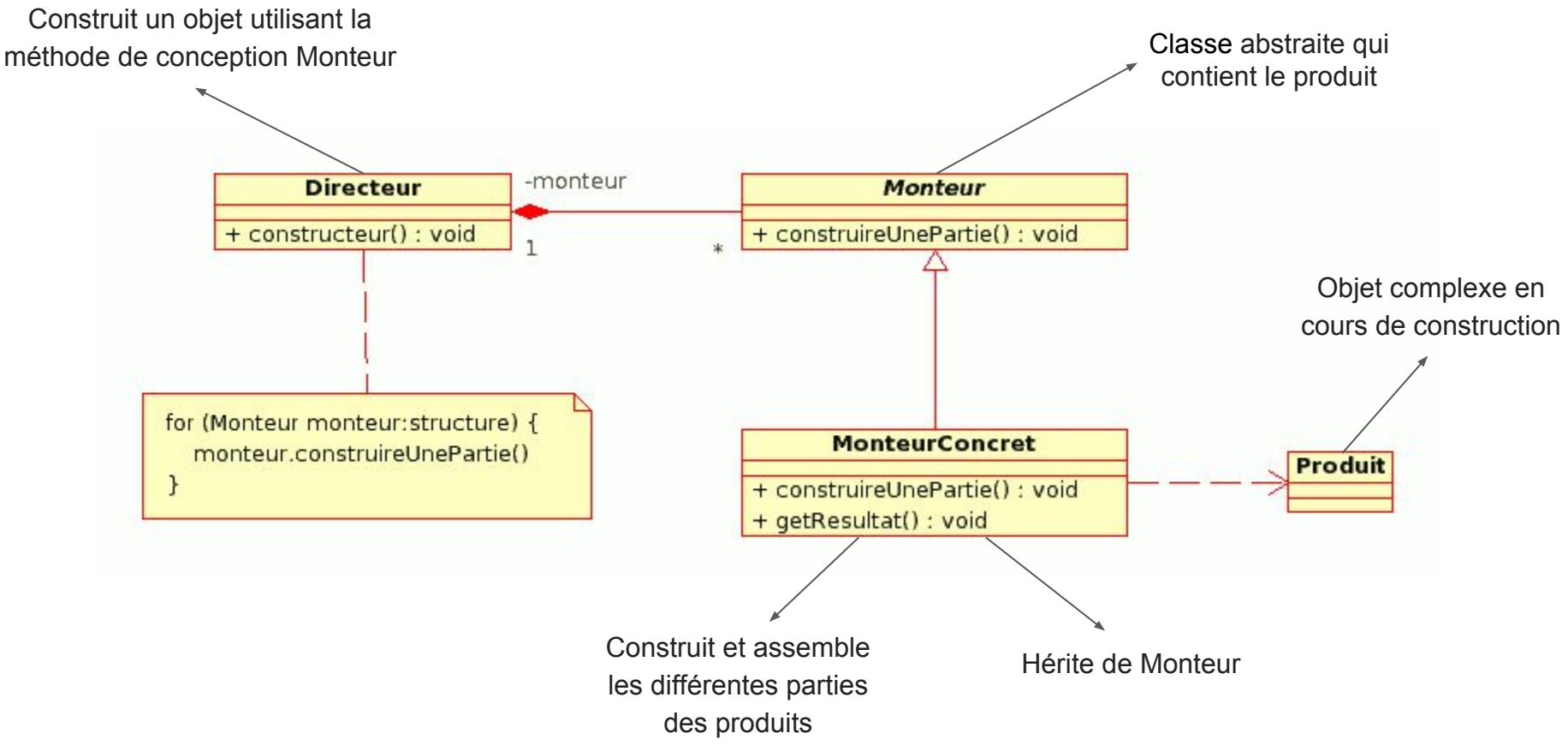
- Nom
- Description du problème à résoudre
- Description de la solution : les éléments de la solution, avec leurs relations
- Conséquences : résultats issus de la solution

Qu’est-ce qu’un Builder Pattern ?

Un objet complexe

Des objets simples

Qu’est-ce qu’un Builder Pattern ?



Qu’est-ce qu’un Builder Pattern ?

Produit

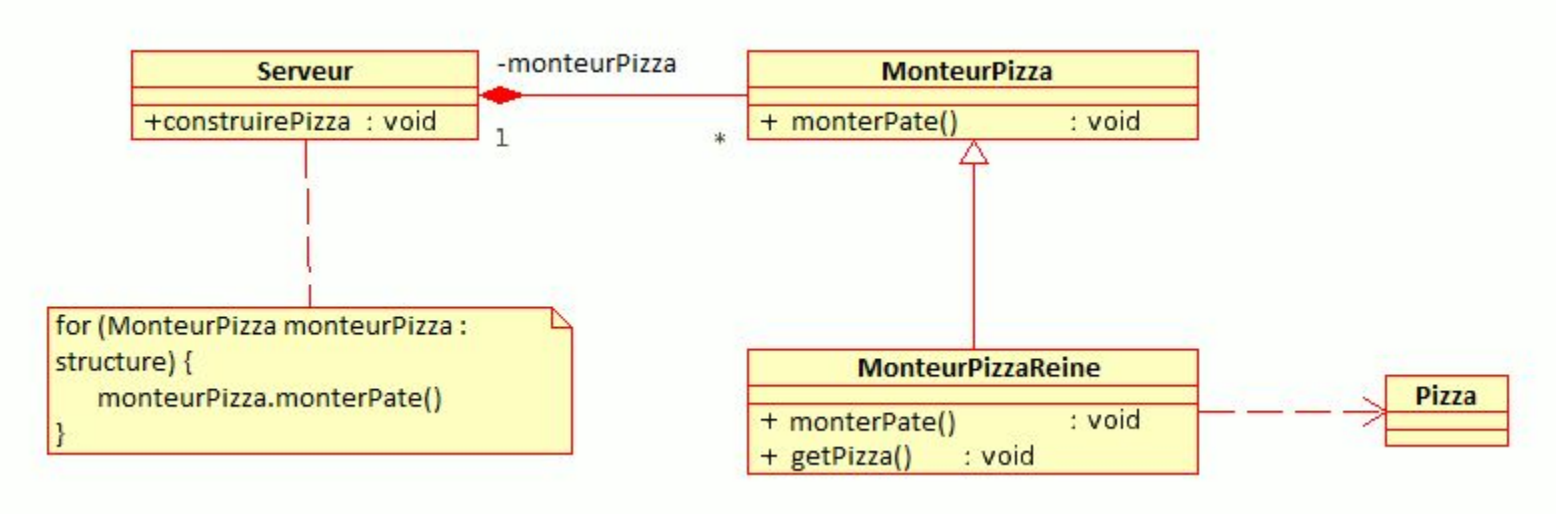
monterPate()

monterSauce()

monterGarniture()

MONTEUR PIZZA

Qu’est-ce qu’un Builder Pattern ?



Avantages du Builder Pattern

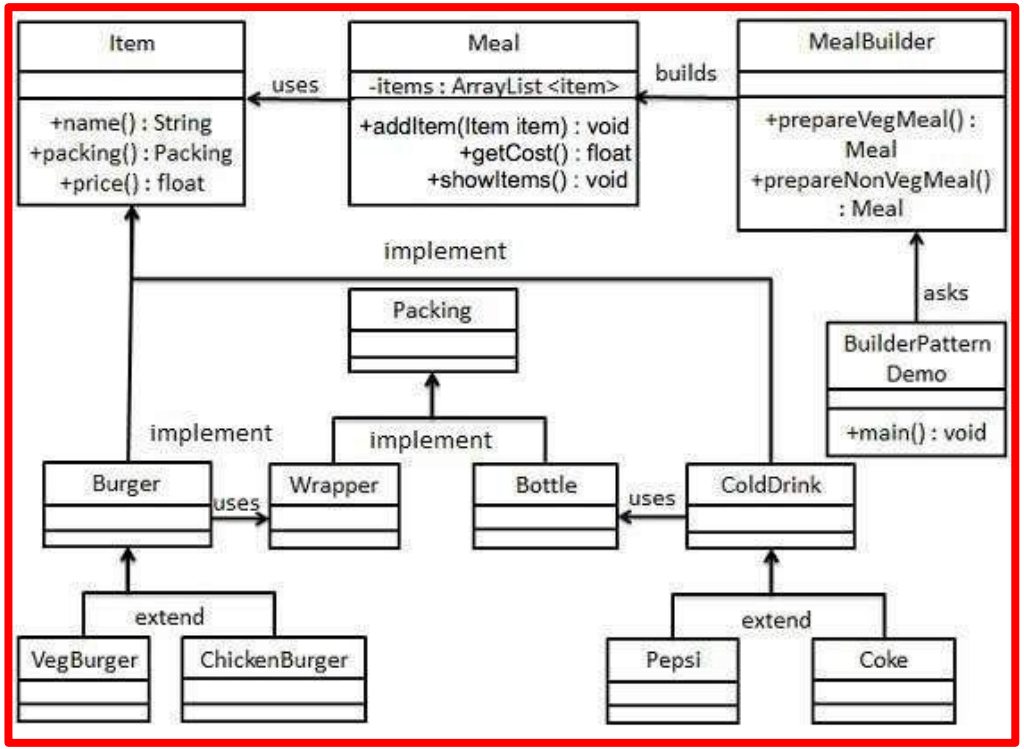
- Séparation claire entre l'ordonnancement de la création, et la construction des différentes parties de l'objet
  → Code plus lisible
- Permet de contrôler les étapes du processus de construction
- Objet est toujours instancié dans un état complet

Inconvénients du Builder Pattern

- Nécessite la création d'un ConcreteBuilder pour chaque type de produit
- Très volumineux
- Ajoute des complexités pas toujours justifiées

Exemple concret d'un Design Pattern Builder

Un menu de fast-food



Etape 1 :

```
public interface Item {
	public String name();
	public Packing packing();
	public float price();
}
```

```
public interface Packing {
	public String pack();
}
```

Etape 2 :

```java
public class Bottle implements Packing {

   @Override
   public String pack() {
      return "Bottle";
   }
}
```

```java
public class Wrapper implements Packing {

   @Override
   public String pack() {
      return "Wrapper";
   }
}
```

Etape 3 :

```
public abstract class Burger implements Item {

   @Override
   public Packing packing() {
      return new Wrapper();
   }

   @Override
   public abstract float price();
}
```

```
public abstract class ColdDrink implements Item {

      @Override
      public Packing packing() {
       return new Bottle();
      }

      @Override
      public abstract float price();
}
```

Etape 4 :

```java
public class ChickenBurger extends Burger {

   @Override
   public float price() {
      return 50.5f;
   }

   @Override
   public String name() {
      return "Chicken Burger";
   }
}
```

```java
public class VegBurger extends Burger {

   @Override
   public float price() {
      return 25.0f;
   }

   @Override
   public String name() {
      return "Veg Burger";
   }
}
```

Etape 4 suite :

```
public class Pepsi extends ColdDrink {

   @Override
   public float price() {
      return 35.0f;
   }

   @Override
   public String name() {
      return "Pepsi";
   }
}
```

```
public class Coke extends ColdDrink {

   @Override
   public float price() {
      return 30.0f;
   }

   @Override
   public String name() {
      return "Coke";
   }
}
```

Etape 5 :

```
import java.util.ArrayList;
import java.util.List;

public class Meal {
   private List<Item> items = new ArrayList<Item>();

   public void addItem(Item item){
      items.add(item);
   }

   public float getCost(){
      float cost = 0.0f;

      for (Item item : items) {
         cost += item.price();
      }
      return cost;
   }
```

```
public void showItems(){

      for (Item item : items) {
         System.out.print("Item : " + item.name());
         System.out.print(", Packing : " +
item.packing().pack());
         System.out.println(", Price : " +
item.price());
      }
   }
}
```

Etape 6 :

```
public class MealBuilder {

   public Meal prepareVegMeal (){
      Meal meal = new Meal();
      meal.addItem(new VegBurger());
      meal.addItem(new Coke());
      return meal;
   }

   public Meal prepareNonVegMeal (){
      Meal meal = new Meal();
      meal.addItem(new ChickenBurger());
      meal.addItem(new Pepsi());
      return meal;
   }
}
```

Etape 7 :

```
public class BuilderPatternDemo {
   public static void main(String[] args) {

      MealBuilder mealBuilder = new MealBuilder();

      Meal vegMeal = mealBuilder.prepareVegMeal();
      System.out.println("Veg Meal");
      vegMeal.showItems();
      System.out.println("Total Cost: " + vegMeal.getCost());

      Meal nonVegMeal = mealBuilder.prepareNonVegMeal();
      System.out.println("\n\nNon-Veg Meal");
      nonVegMeal.showItems();
      System.out.println("Total Cost: " + nonVegMeal.getCost());
   }
}
```

Résultat :

```
Veg Meal
Item : Veg Burger, Packing : Wrapper, Price : 25.0
Item : Coke, Packing : Bottle, Price : 30.0
Total Cost: 55.0


Non-Veg Meal
Item : Chicken Burger, Packing : Wrapper, Price : 50.5
Item : Pepsi, Packing : Bottle, Price : 35.0
Total Cost: 85.5
```

Sources :

https://www.tutorialspoint.com/design_pattern/builder_pattern.htm
https://fr.wikipedia.org/wiki/Patron_de_conception
https://fr.wikipedia.org/wiki/Pattern
https://sourcemaking.com/design_patterns
https://jormes.developpez.com/articles/design-pattern-construction/
https://blog.xebia.fr/2016/12/28/design-pattern-builder-et-builder-sont-dans-un-bateau/
https://fr.wikipedia.org/wiki/Monteur_(patron_de_conception)
https://en.wikipedia.org/wiki/Builder_pattern
https://www.geeksforgeeks.org/builder-design-pattern/
https://www.tutorialspoint.com/design_pattern/builder_pattern.htm