Adam - Louis - Jules - Sébastien

Builder.

SOMMAIRE

Design Patterns: Elements of Reusable Object-Oriented Software

Gang of Four (GoF) :

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides.

CREATION

STRUCTURE

COMPORTEMENT

SINGLETON

un seul

BUILDER

LE PROBLEME

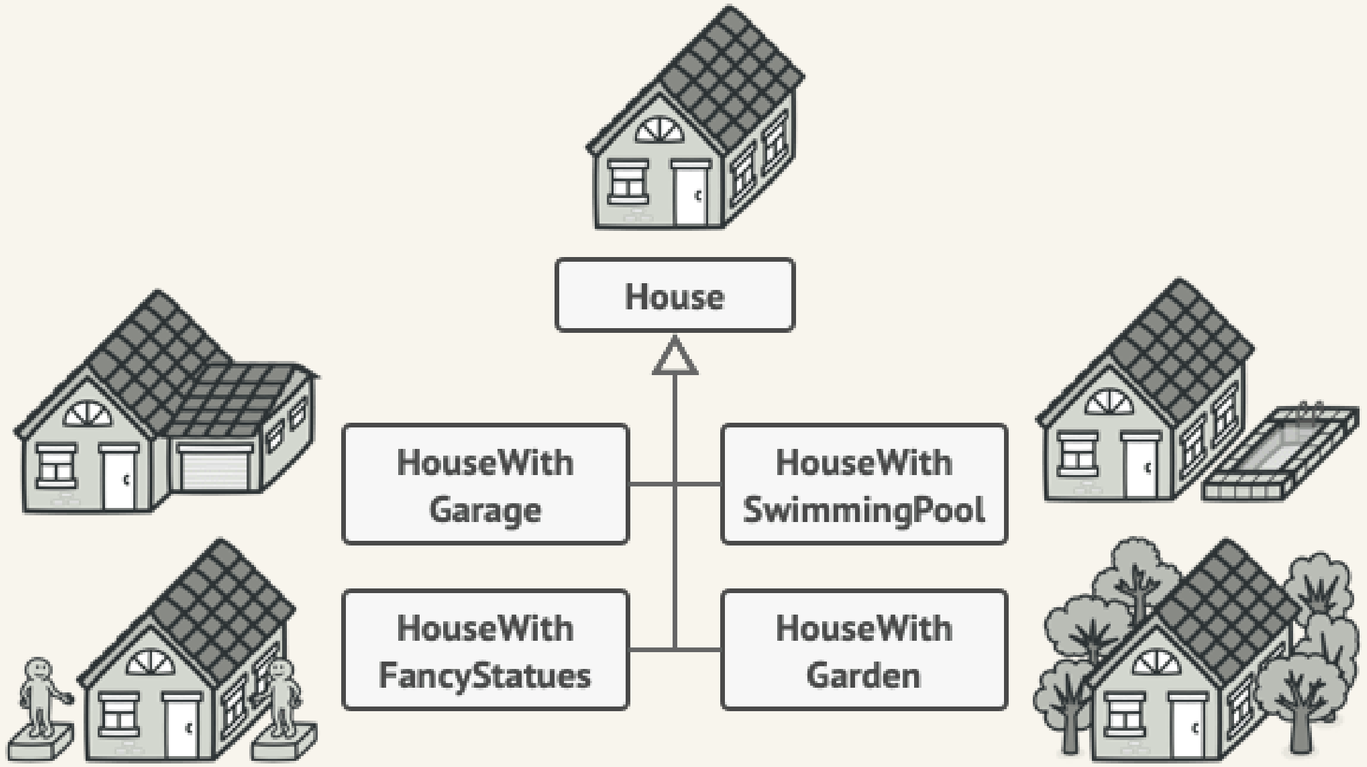
Personnage

- Sting nom;
- date anniversaire ;
- couleur de cheveux ;
- taille ;
- classe ;
- force ;
- défense ;
- inventaire ;
- etc.

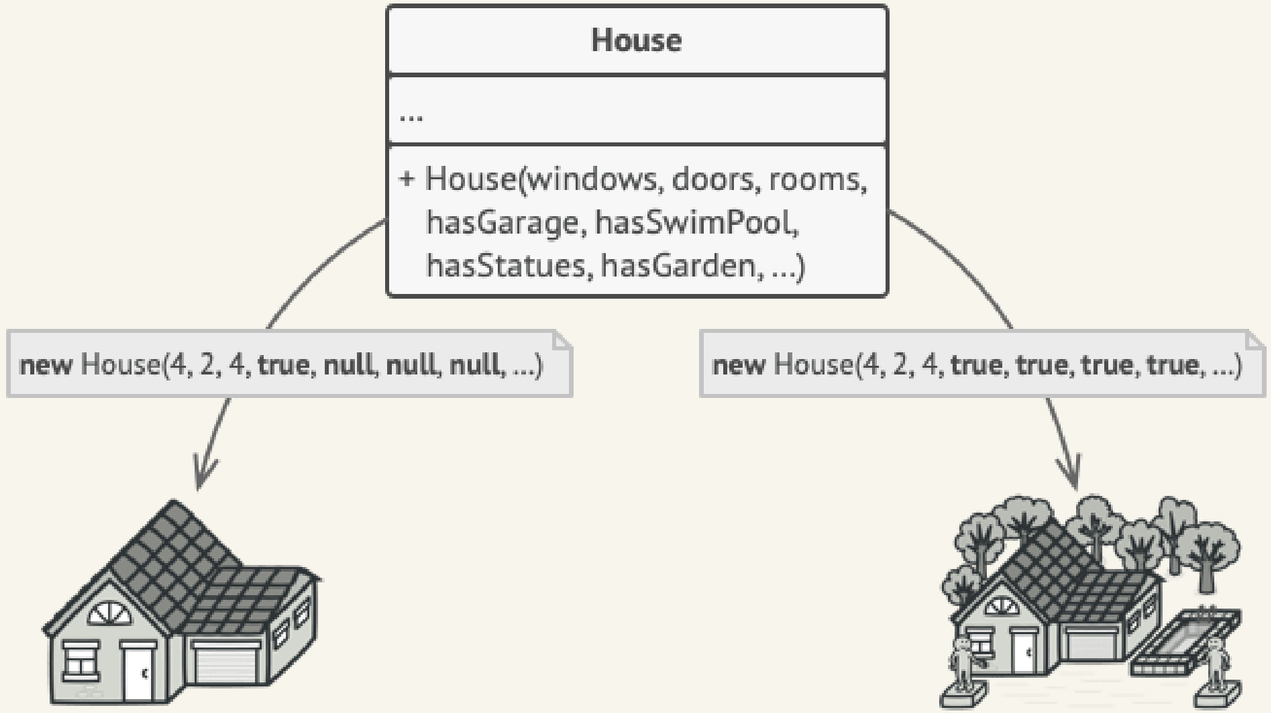
+Gagner un niveau()
+ etc ()

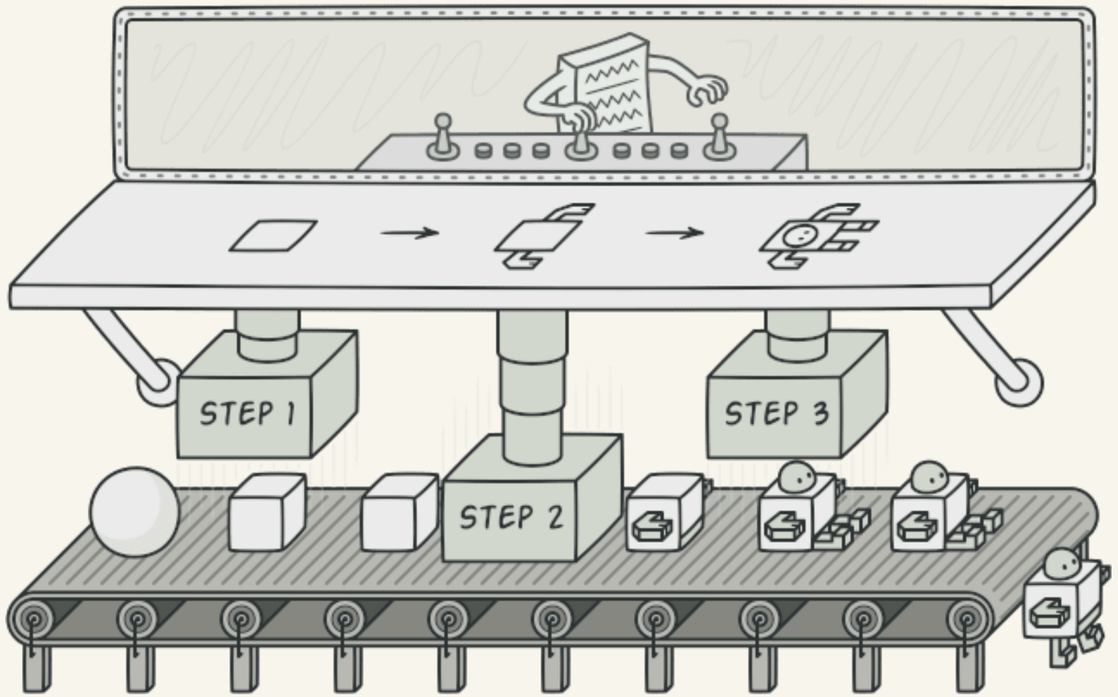
```java
public Personnage(String nom, Date anniversaire, Couleur
                  couleurDeCheveux, int taille, Classe classe,
                  int force, int defense, Inventaire inventaire) {
    this.nom = nom;
    this.anniversaire = anniversaire;
    this.couleurDeCheveux = couleurDeCheveux;
    this.taille = taille;
    this.classe = classe;
    this.force = force;
    this.defense = defense;
    this.inventaire = inventaire;
}

public Personnage(String nom, Date anniversaire, Couleur
                  couleurDeCheveux, int taille) {
    this.nom = nom;
    this.anniversaire = anniversaire;
    this.couleurDeCheveux = couleurDeCheveux;
    this.taille = taille;
    this.classe = Classe.DEFAULT;
    this.force = 0;
    this.defense = 0;
    this.inventaire = new Inventaire();
}
```

Sans builder :



Sans builder :

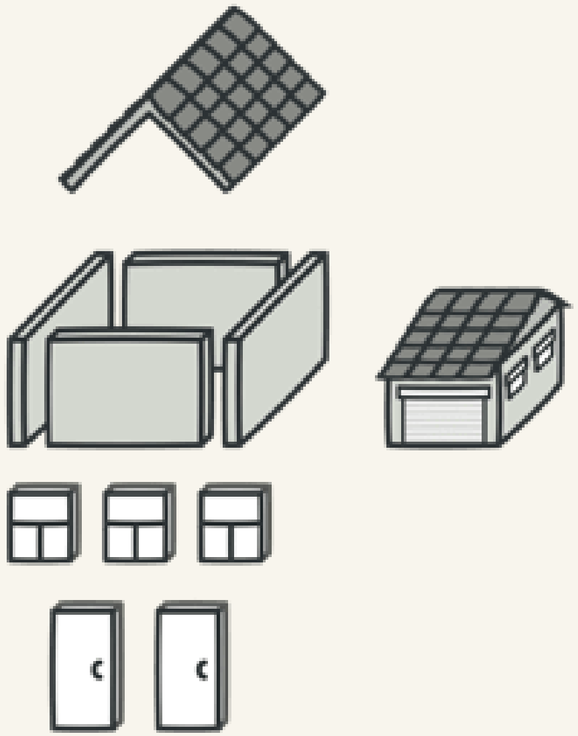




HouseBuilder

...

+ buildWalls()
+ buildDoors()
+ buildWindows()
+ buildRoof()
+ buildGarage()
+ getResult(): House



Solution avec builder

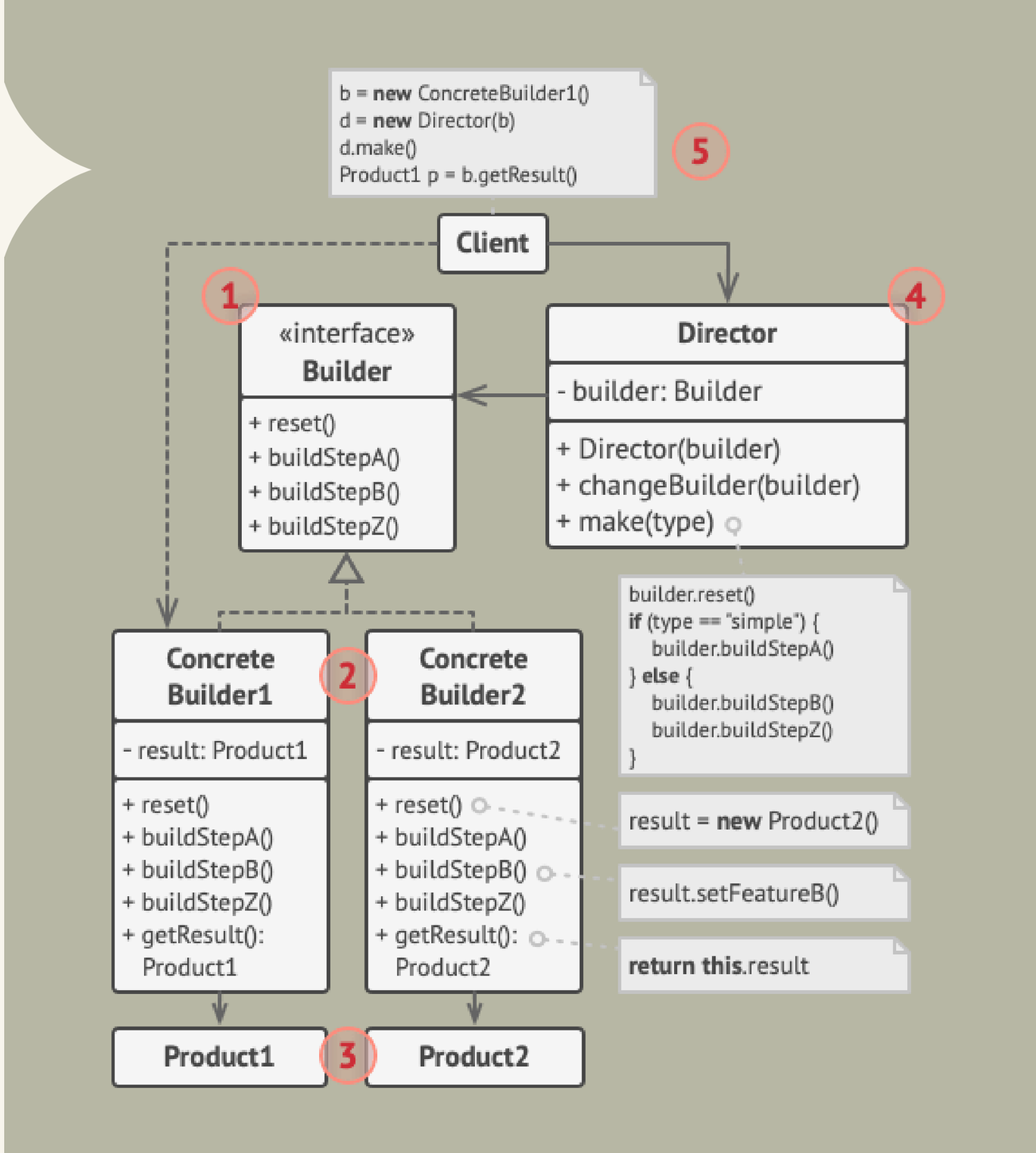
Builder (GoF)

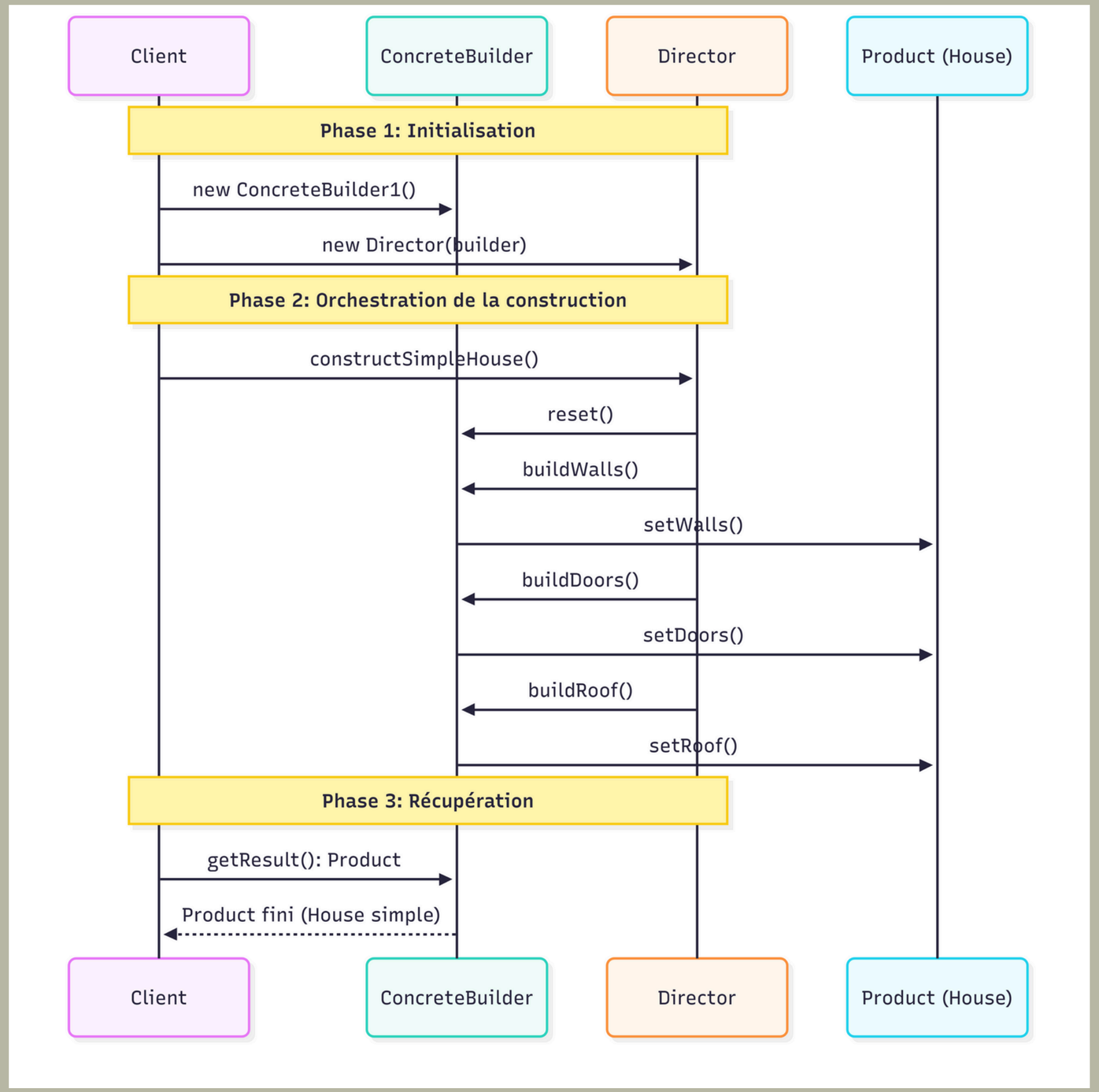
Product
→ l'objet final

Builder
→ définit les étapes de construction

ConcreteBuilder
→ implémente les étapes concrètes

Director
→ contrôle l'ordre de construction




Client
ConcreteBuilder
Director
Product (House)
Phase 1: Initialisation
new ConcreteBuilder1()
new Director(builder)
Phase 2: Orchestration de la construction
constructSimpleHouse()
reset()
buildWalls()
setWalls()
buildDoors()
setDoors()
buildRoof()
setRoof()
Phase 3: Récupération
getResult(): Product
Product fini (House simple)
Client
ConcreteBuilder
Director
Product (House)

SOLID

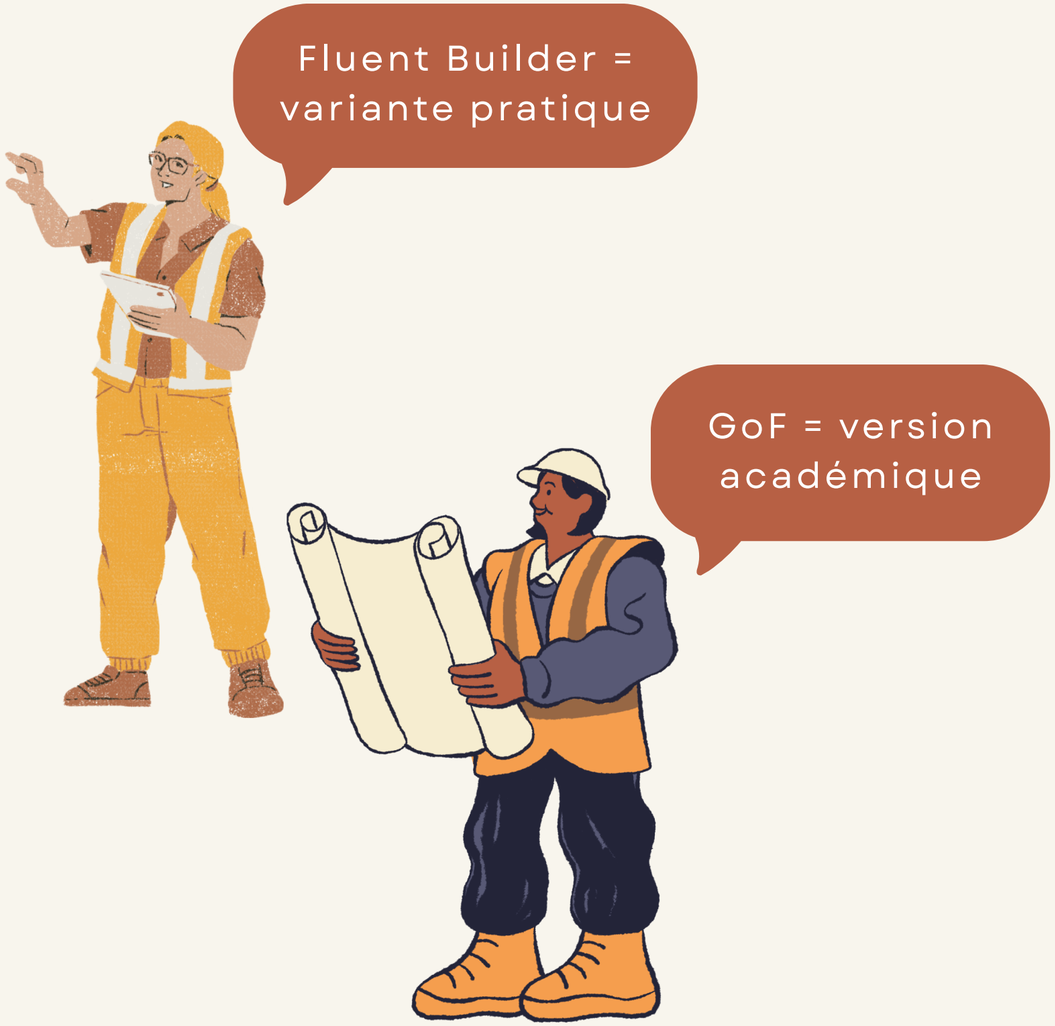
- **S**RP :
  Dans le Builder, chaque rôle a sa responsabilité
- **O**CP :
  Ajout de nouvelle classe sans modifier les autres
- **L**
- **I**
- **D**IP :
  Le Director dépend de l’interface Builder

Les limites

- Pas utile pour des objets simples.
- Peut rendre la conception plus lourde.

```
//Avec un constructeur
User u = new User("Alice", "1234");

//Avec un Builder
User v = new UserBuilder()  no usages
        .setUsername("Alice")
        .setPassword("1234")
        .build();
```

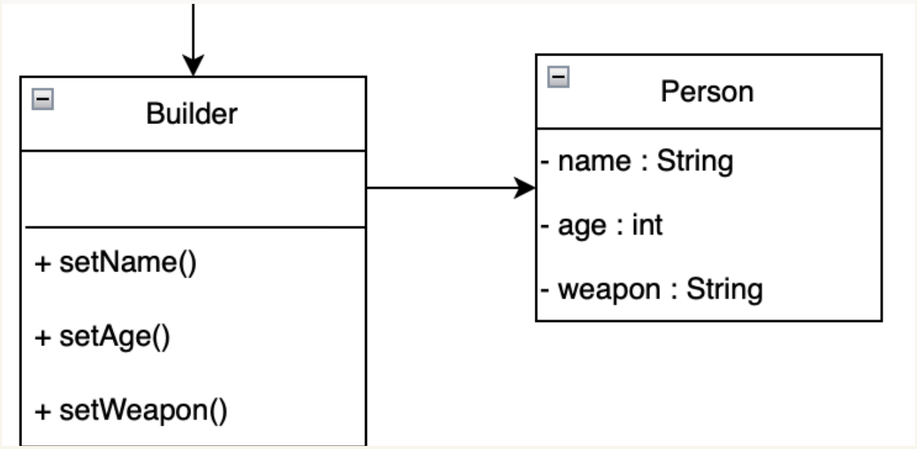
Varianies & Comparaisons du Builder



Flueni Builder

- Pas besoin de Director
- Plus lisible et pratique

```
Person p = new Person.Builder()
        .setName("Arthur")
        .setAge(25)
        .setWeapon("Sword")
        .build();
```

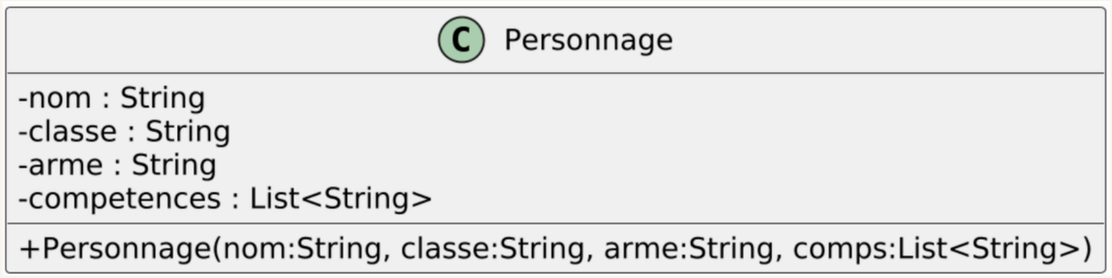


Comparaison avec d'autres patterns de création

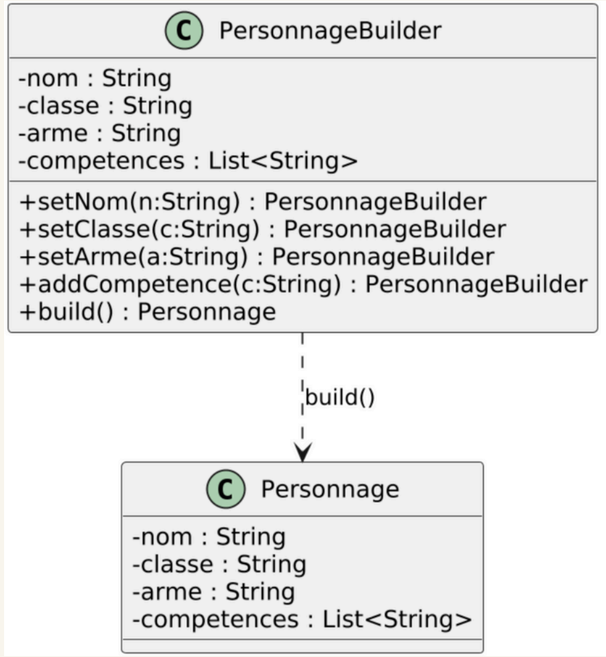
Pattern	Objectif principal	Exemple concret
Factory Method	Créer un seul objet spécifique	Une arme unique
Abstract Factory	Créer une famille d'objets liés	Interface GUI (bouton + menu)
Builder	**Construire un objet complexe étape par étape**	**Personnage de jeu**

Création d'un personnage de jeu vidéo

- Sans Builder :
  - Constructeur avec trop de paramètres
  - Difficile à lire et à modifier



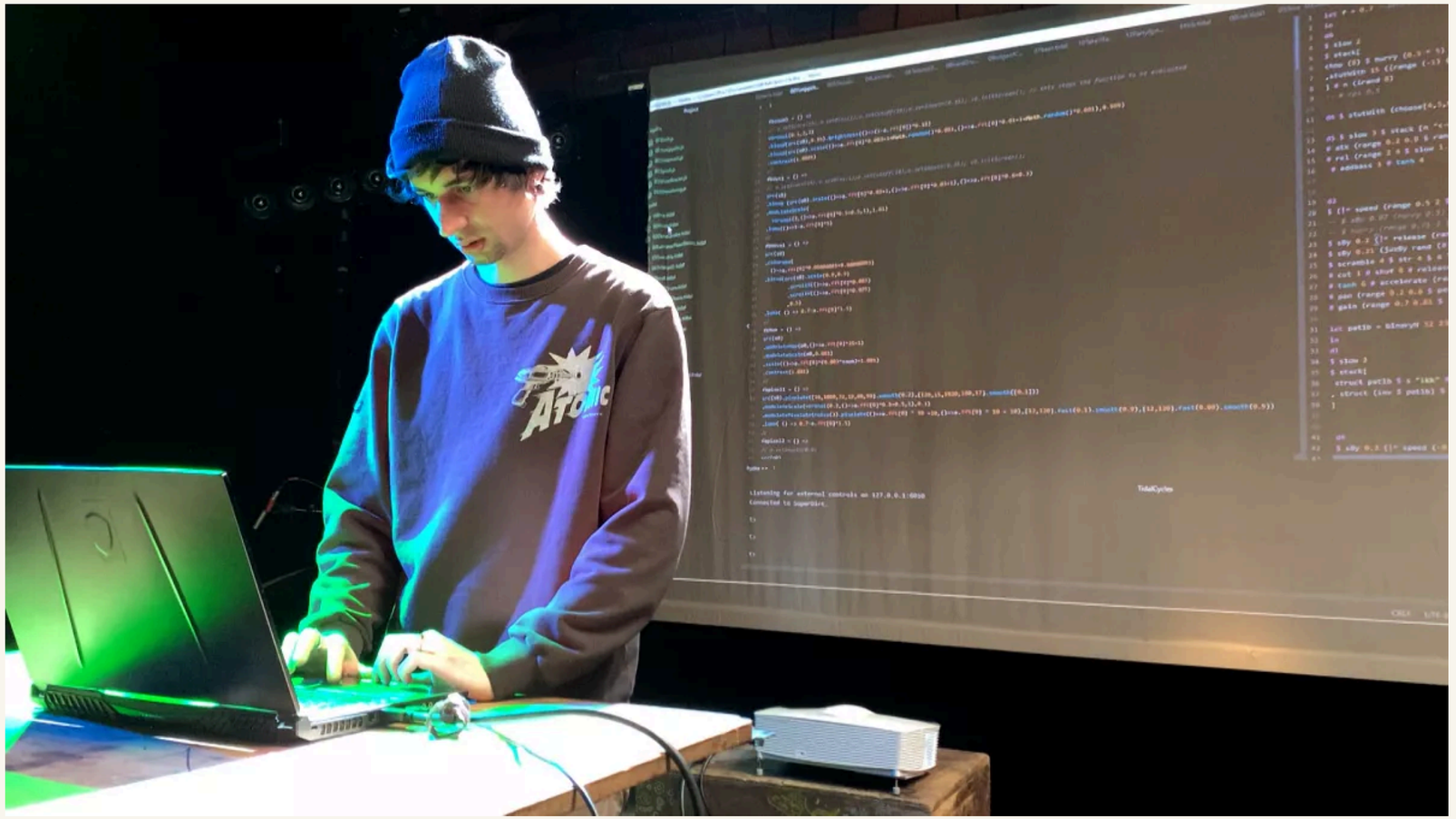
- Avec Builder :
  - Construction progressive
  - Code plus clair et lisible

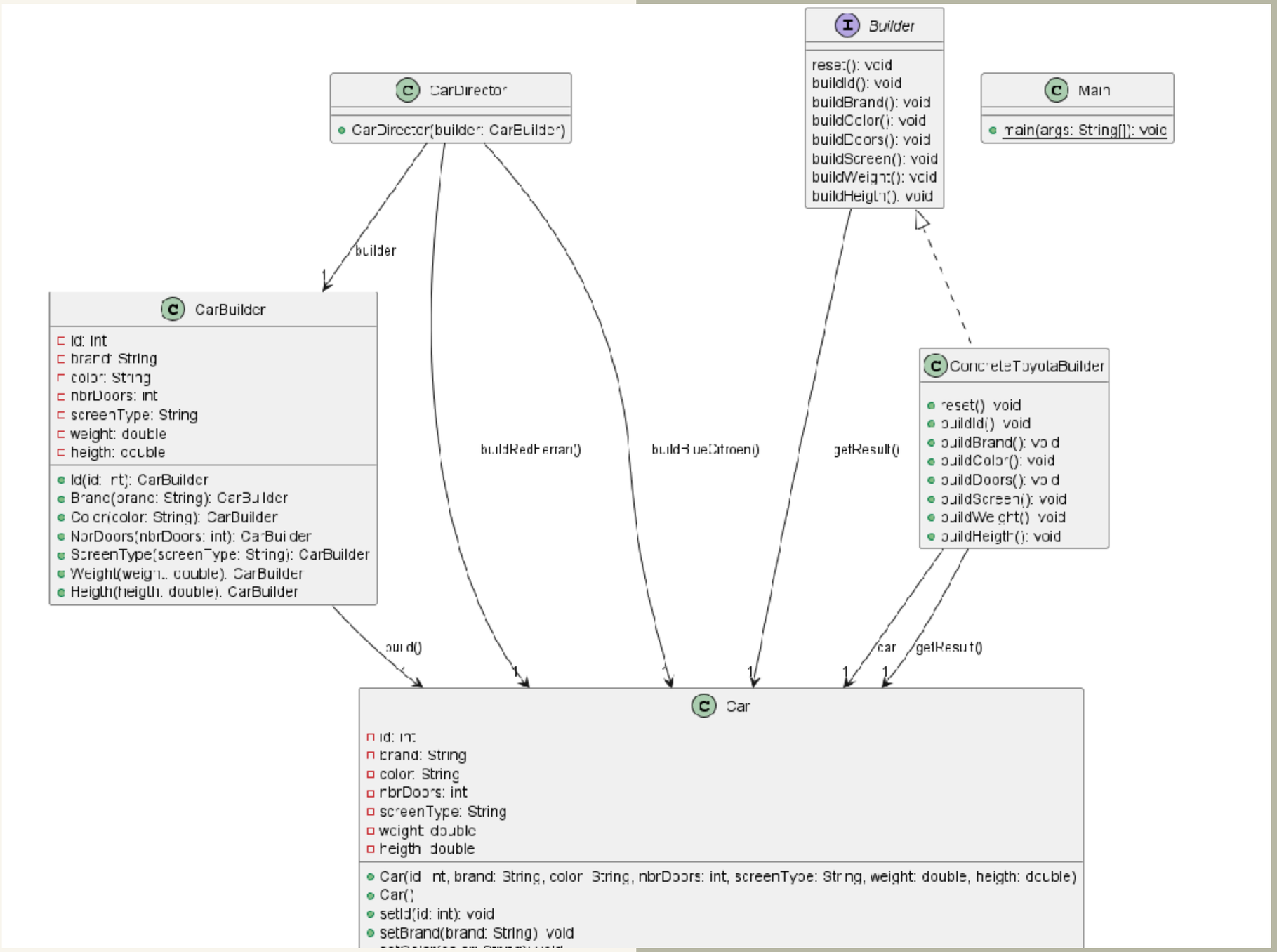


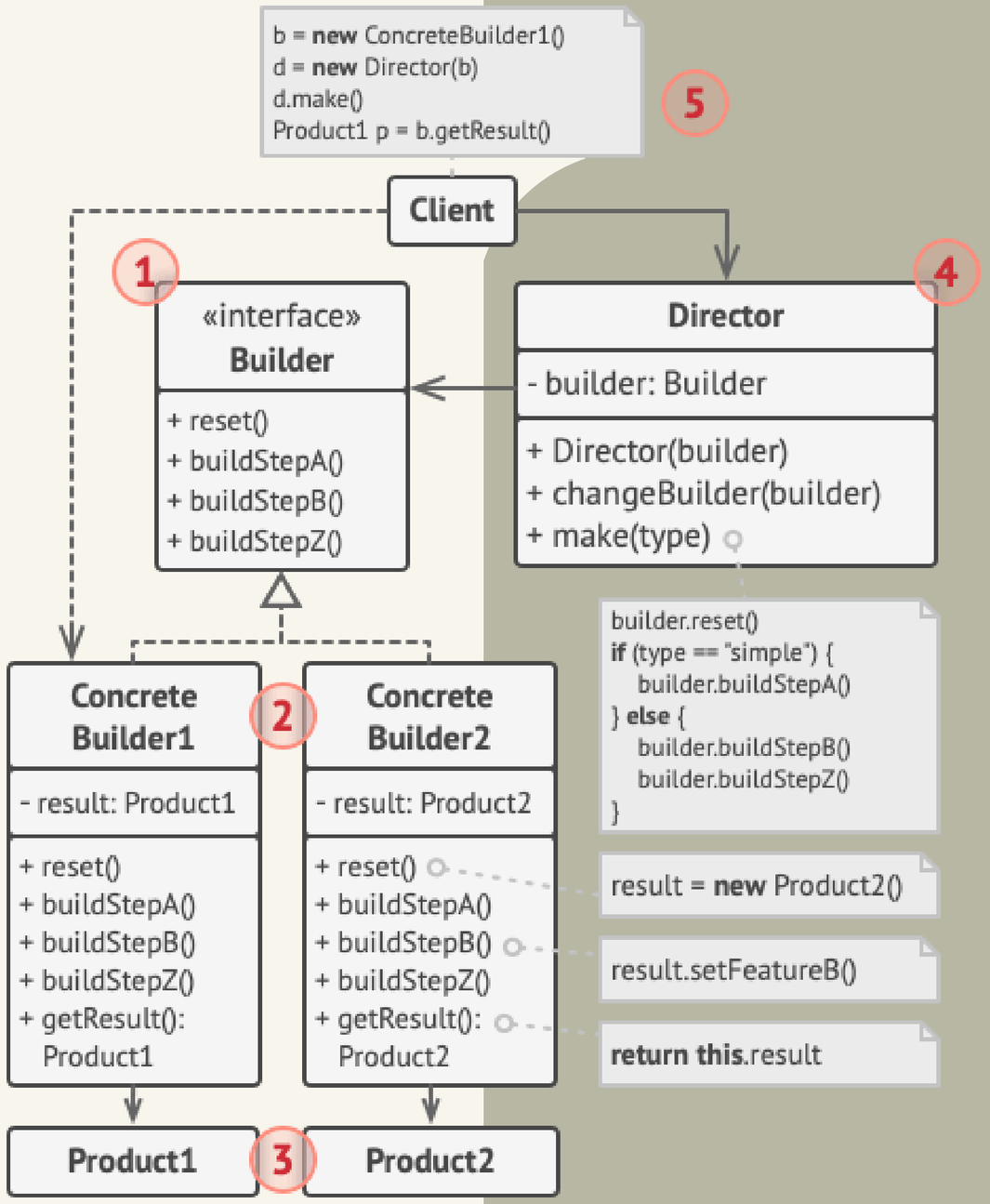
Exemple appel :

```
Personnage p = new Personnage.Builder()
    .setNom("Arthur")
    .setClasse("Guerrier")
    .setArme("Épée")
    .addCompetence("Force")
    .build();
```

Live-coding




CarDirector
+ CarDirector(builder: CarBuilder)
builder
Builder
reset(): void
buildId(): void
buildBrand(): void
buildColor(): void
buildDoors(): void
buildScreen(): void
buildWeight(): void
buildHeigth(): void
Main
+ main(args: String[]): void
CarBuilder
- id: int
- brand: String
- color: String
- nbrDoors: int
- screenType: String
- weight: double
- heigth: double
+ Id(id: int): CarBuilder
+ Brand(brand: String): CarBuilder
+ Color(color: String): CarBuilder
+ NbrDoors(nbrDoors: int): CarBuilder
+ ScreenType(screenType: String): CarBuilder
+ Weight(weight: double): CarBuilder
+ Heigth(heigth: double): CarBuilder
buildRedFerrari()
buildBlueCitroen()
getResult()
ConcreteToyotaBuilder
+ reset(): void
+ buildId(): void
+ buildBrand(): void
+ buildColor(): void
+ buildDoors(): void
+ buildScreen(): void
+ buildWeight(): void
+ buildHeigth(): void
build()
car
getResult()
Car
- id: int
- brand: String
- color: String
- nbrDoors: int
- screenType: String
- weight: double
- heigth: double
+ Car(id: int, brand: String, color: String, nbrDoors: int, screenType: String, weight: double, heigth: double)
+ Car()
+ setId(id: int): void
+ setBrand(brand: String): void


b = new ConcreteBuilder1()
d = new Director(b)
d.make()
Product1 p = b.getResult()
5
Client
1
«interface»
Builder
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
4
Director
- builder: Builder
+ Director(builder)
+ changeBuilder(builder)
+ make(type)
builder.reset()
if (type == "simple") {
builder.buildStepA()
} else {
builder.buildStepB()
builder.buildStepZ()
}
Concrete
Builder1
2
- result: Product1
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product1
Concrete
Builder2
- result: Product2
+ reset()
+ buildStepA()
+ buildStepB()
+ buildStepZ()
+ getResult():
Product2
result = new Product2()
result.setFeatureB()
return this.result
Product1
3
Product2

QCM

Bibliographie

- https://www.youtube.com/watch?v=MaY_MDdWkQw&t=1s
- https://refactoring.guru
- https://www.youtube.com/watch?v=vzHMeMsjOcw
- https://www.youtube.com/watch?v=KbIdk5BRn0w
- https://medium.com/@faroukymedia/le-pattern-builder-d1ed9a65f6a3

Merci de nous avoir écoutés