

DESIGN PATTERNS

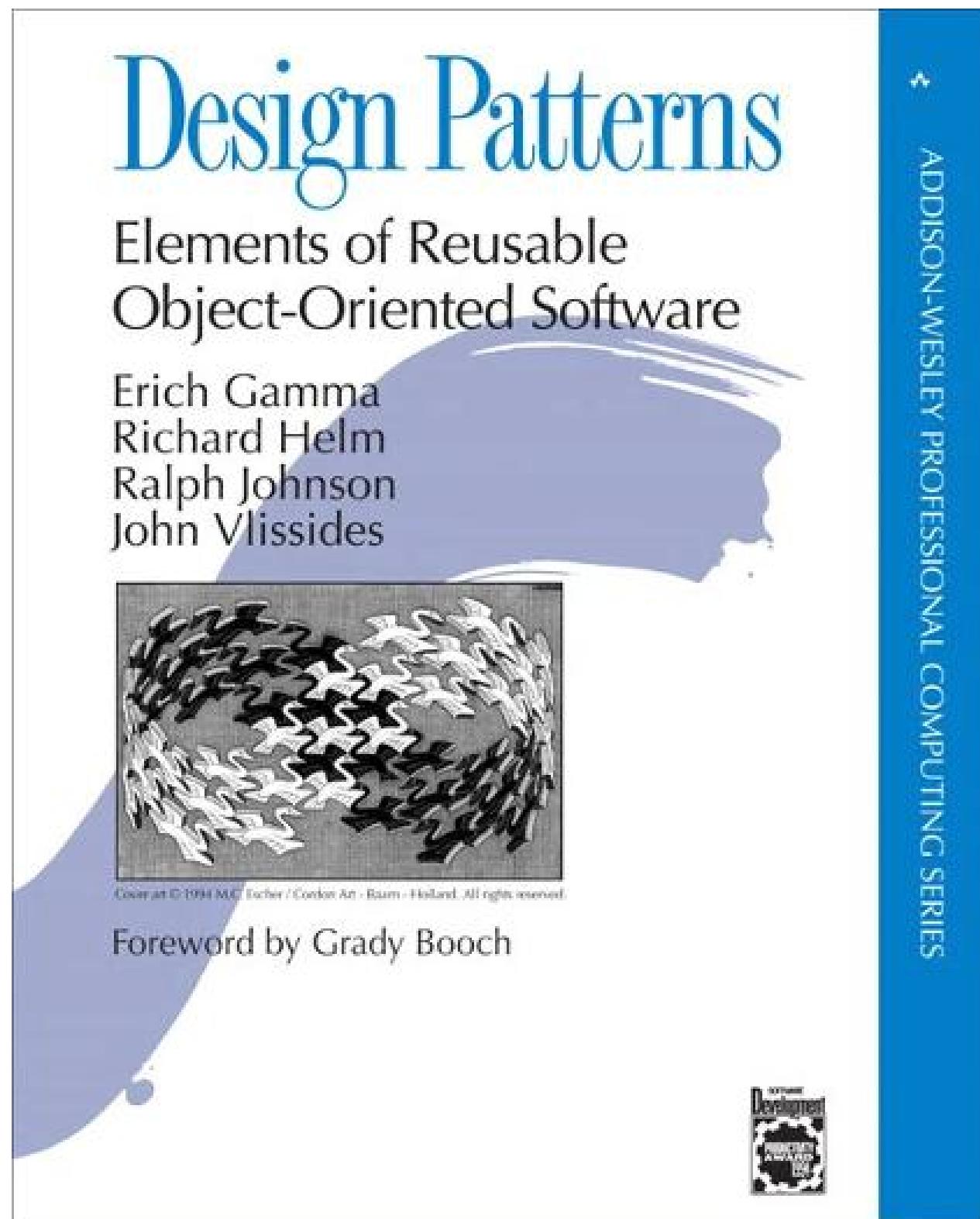
# BUILDER



# Le pattern BUILDER

## Appartient aux patterns de création

GOF / Gang of Four

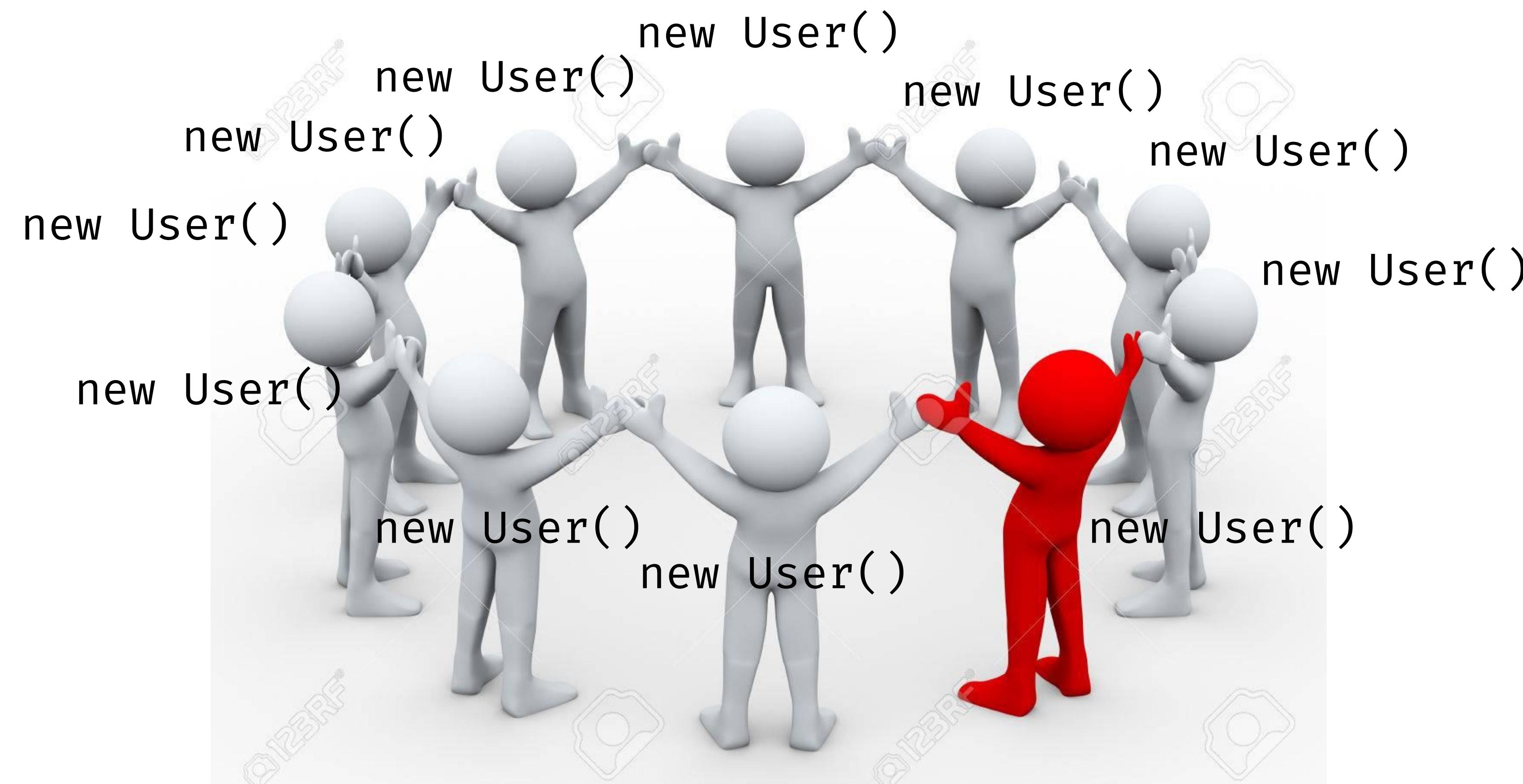


terminally online engineer 🇺🇦 ✅ @tekborg · 8 févr.

>java dev suggests to implement some **pattern**  
>ask him if it's a good **pattern** or some wizardry  
>he doesn't understand  
>pull up a diagram explaining useless layers of abstractions  
>he laughs and says it's a good **pattern** sir  
>look it up  
>it's some wizardry

A photograph of Steve Jobs, co-founder of Apple. He is wearing a dark blue zip-up vest over a red and blue plaid shirt. He has his hands on his hips and is looking directly at the camera with a serious expression. The background is slightly blurred, showing other people in what appears to be a public event or press conference.

Exemple : Il nous est demandé de créer des utilisateurs pour une raison quelconque raison.



## User

- identifier: String
- firstName: String
- lastName: String
- isAdministrator: Boolean
- groups: Group[]

+User(identifier: String,  
firstName: String, lastName:  
String, isAdministrator:  
Boolean, groups: Group[])

+getIdentifier(): String

+getFirstName(): String

+getLastName(): String

+isAdministrator(): Boolean

+groups(): Group[]

# Première modélisation

```
new User("doe1", "John", "DOE", false, []);  
  
new User("unilimadmin", "Unilim", "ADMIN", true, [  
    new Group( ... ),  
    new Group( ... )  
]);  
  
new User("ultest", "Persona", "AUTRE", false, []);
```

Un exemple d'utilisation en TypeScript...

## User

```
- identifier: String  
- firstName: String  
- lastName: String  
- isAdministrator: Boolean  
- groups: Group[]
```

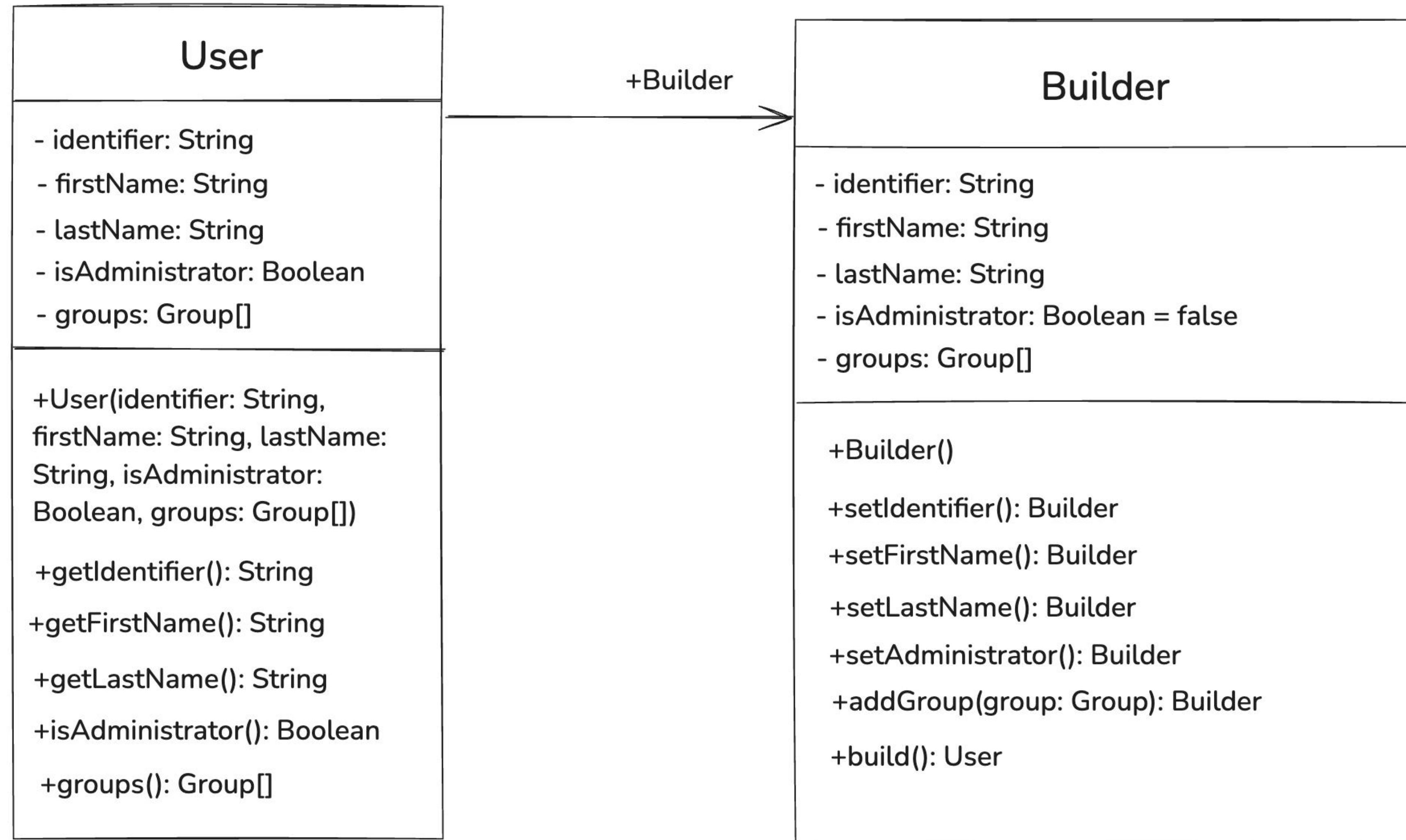
```
+User(identifier: String,  
firstName: String, lastName:  
String, isAdministrator:  
Boolean, groups: Group[])
```

```
+getIdentifier(): String  
+getFirstName(): String  
+getLastName(): String  
+isAdministrator(): Boolean  
+groups(): Group[]
```

- Trop de paramètres dans le constructeur (anti-pattern)
- Complexe à maintenir

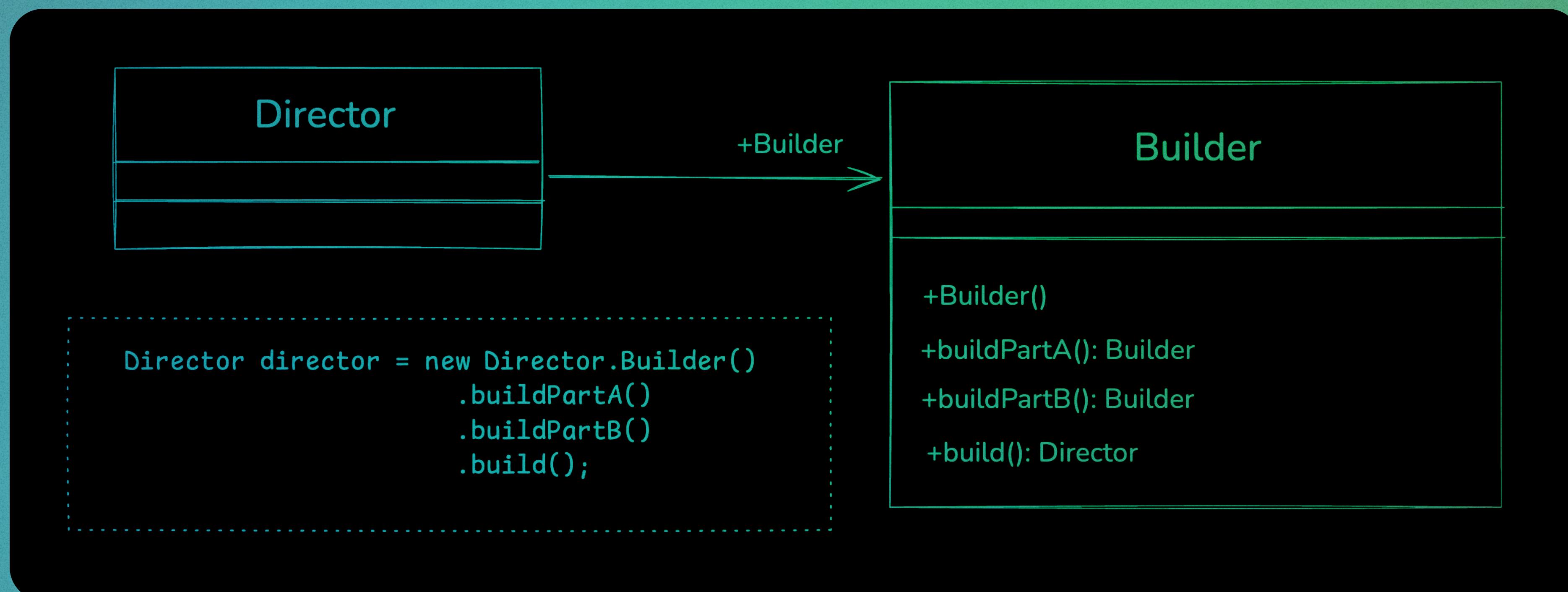


# Une solution avec le pattern **BUILDER**



# CAS GÉNÉRAL

Parfois, une interface peut être ajoutée pour avoir plusieurs builders.



# Quand l'utiliser ?

- Quand l'algorithme de création doit être indépendant
- Quand le processus de construction doit permettre différentes représentations de l'objet construit
- Quand il y a une séquence spécifique lors de la création d'un objet

# S O L I D

## SRP

(Single Responsibility Principle)

Une classe doit avoir **une seule responsabilité**.

Dans le cas d'objets complexes, toutes les étapes sont confiés au builder, **la classe garde donc sa propre responsabilité**.



## OCP

(Open/Closed Principle)

Une classe doit être ouverte aux extensions **sans modifier le code existant**.

On peut rajouter des options aux builders **sans avoir à modifier ce qui est déjà fait**.

# Mais il y a des limites...

## Augmentation de la complexité du code

Ajout d'une classe en plus.

## Sur-utilisation possible

Aucun intérêt si le constructeur initial n'est pas complexe.

## Diminution des performances

On crée plusieurs objets pour en faire un autre.



# Utilité dans les tests

```
void total_sum_should_be_calculated() {  
    // Arrange  
  
    // Act  
    int total = order.getTotal();  
  
    // Assert  
    assertEquals(50, total);  
}
```

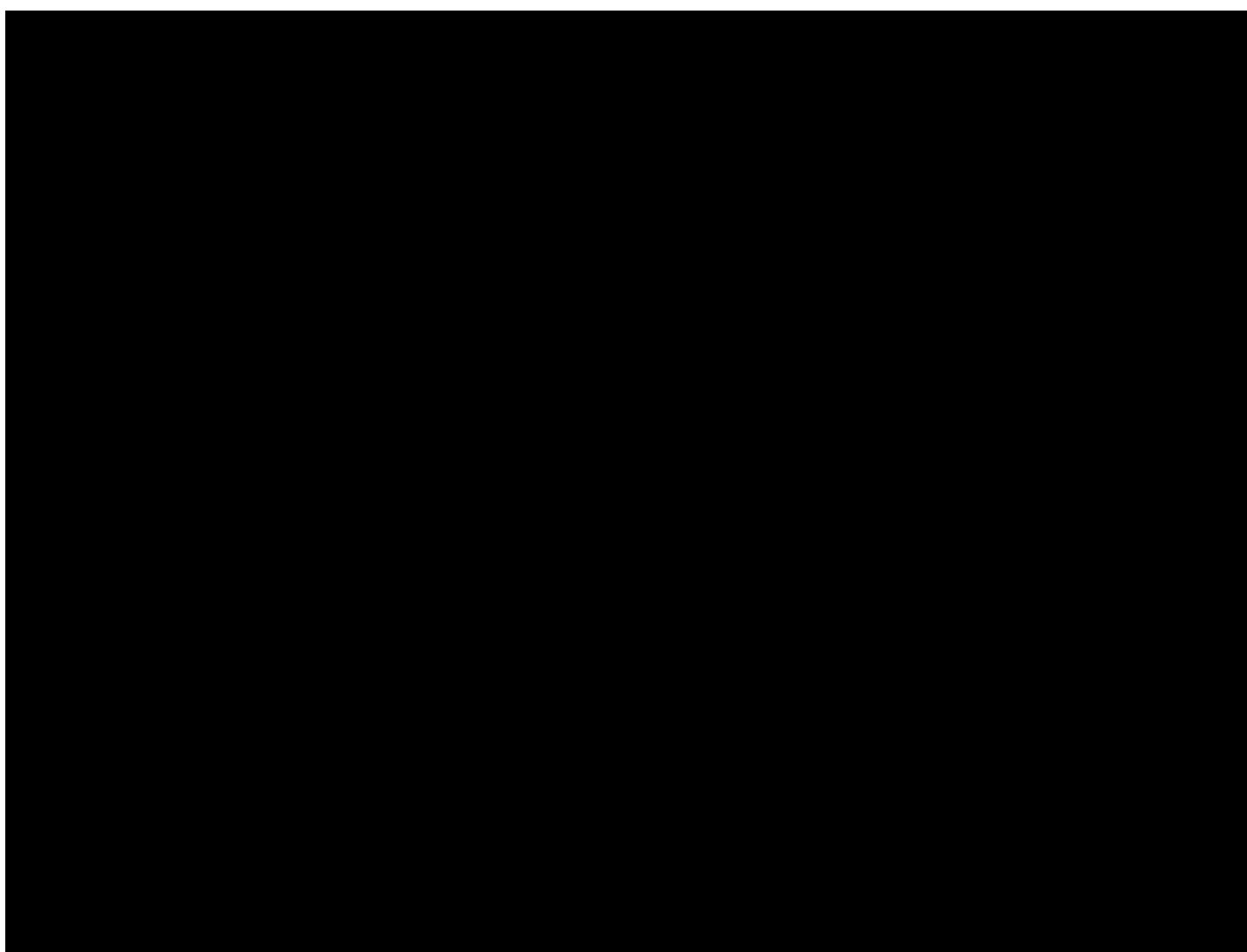


```
Order order = new Order(  
    null, null, null,  
    null, null, null,  
    null, null, null,  
    null, DateTime.Parse("2025-01-18"));  
  
order.addOrderItem(new OrderItem(  
    1,  
    10,  
    null,  
    null,  
    null,  
    null));  
  
order.addOrderItem(new OrderItem(  
    2,  
    15,  
    null,  
    null,  
    null,  
    null));
```



# Utilité dans les tests

```
void total_sum_should_be_calculated() {  
    // Arrange
```



```
Order order = new OrderBuilder()  
    .setOrderDate(DateTime.Parse("2020-01-18"))  
    .addOrderItem(new OrderItemBuilder()  
        .setId(1)  
        .setUnitPrice(10)  
        .build())  
    .addOrderItem(new OrderItemBuilder()  
        .setId(2)  
        .setUnitPrice(15)  
        .build())  
    .addOrderItem(new OrderItemBuilder()  
        .setId(3)  
        .setUnitPrice(25)  
        .build())  
    .build();
```

```
// Act  
int total = order.getTotal();  
  
// Assert  
assertEquals(50, total);  
}
```



LIVE E. CODE

# Bibliographie

<https://medium.com/@iamprovidence/unit-tests-and-builder-pattern-%D0%B0-match-made-in-heaven-f9ff25a3befa>

<https://java-design-patterns.com/patterns/builder/#real-world-applications-of-builder-pattern-in-java>

<https://www.vogella.com/tutorials/DesignPatternBuilder/article.html>

<https://bgasparotto.com/design-patterns/builder>

[https://en.wikipedia.org/wiki/Builder\\_pattern](https://en.wikipedia.org/wiki/Builder_pattern)

