

Voici la traduction en français de l'article « **26 Reasons Why Using Optional Correctly Is Not Optional** » publié sur DZone par Anghel Leonard [DZone](#).

26 raisons pour lesquelles utiliser `Optional` correctement n'est pas facultatif

Introduction

Les `Optional` ne sont pas aussi simples qu'ils en ont l'air. Selon Brian Goetz, l'architecte du langage Java :

"Optional est destiné à fournir un mécanisme limité pour les types de retour des méthodes de bibliothèque quand il est nécessaire de représenter « aucun résultat », et l'utiliser à la place de `null` risquait fortement de provoquer des erreurs." [DZone](#)

Pour bien utiliser `Optional`, abordons 26 points clés — chacun illustrant une bonne ou mauvaise pratique.

Item 1 : Ne jamais assigner `null` à une variable `Optional`

À éviter :

```
Optional<Cart> emptyCart = null;
```

À préférer :

```
Optional<Cart> emptyCart = Optional.empty();
```

Item 2 : Vérifier la présence avant d'utiliser `Optional.get()`

À éviter :

```
Optional<Cart> cart = ...;  
Cart myCart = cart.get(); // peut lancer NoSuchElementException
```

À préférer :

```
if (cart.isPresent()) {  
    Cart myCart = cart.get();  
    // ...  
} else {  
    // ...  
}
```

Item 3 : Utiliser `orElse()` pour renvoyer un objet par défaut déjà construit

Attention au coût de performance car l'argument est évalué même si non utilisé.

Item 4 : Pour des créations coûteuses, utiliser `orElseGet()` avec un `Supplier`

Item 5 : Lancer une exception `NoSuchElementException` avec `orElseThrow()` (Java 10+)

Item 6 : Pour une exception explicite, utiliser `orElseThrow(Supplier<...>)`

Item 7 : Lorsque nécessaire, récupérer une référence pouvant être `null` via `orElse(null)`

Item 8 : Utiliser `ifPresent()` pour consommer la valeur si elle est présente

Item 9 : Avec Java 9+, préférer `ifPresentOrElse()` pour gérer aussi le cas "absent"

Item 10 : Pour renvoyer un autre `Optional` si vide, utiliser `or()` (Java 9+)

Item 11 : Les méthodes `orElse()` / `orElseGet()` sont idéales pour chaîner dans des lambdas

Item 12 : Ne pas chaîner `Optional` uniquement pour obtenir une valeur — dans certains cas, un simple `null` check est plus clair

Item 13 à 16 : Ne pas utiliser `Optional` pour :

- champs de classe (fields)

- arguments de constructeur
 - arguments de méthode
- Mieux vaut gérer la présence avec des surcharges de méthodes ou des arguments simples [DZone+2DZone+2](#).
-

Item 17 : Évitez de renvoyer des collections vides enveloppées dans `Optional`

Privilégiez plutôt les collections vides directement [DZone](#).

Item 18 : Ne pas utiliser `Optional` dans les collections — cela complique le design et pénalise les performances [DZone](#).

Item 19 : Distinction entre `Optional.of()` et `Optional.ofNullable()`

- `Optional.of(null)` → `NullPointerException`
 - `Optional.ofNullable(null)` → `Optional.empty()`
-

Item 20 : Pour les types primitifs, préférez `OptionalInt`, `OptionalLong` ou `OptionalDouble` afin d'éviter les boxing coûts [DZone](#).

Item 21 : Pas besoin de débiller pour comparer deux `Optional` dans un test — `equals()` compare les valeurs internes [DZone](#).

Item 22 : Utiliser `map()` et `flatMap()` pour transformer élégamment des valeurs `Optional`

Item 23 : `filter()` permet de rejeter une valeur selon un prédicat, sans déballage explicite [DZone](#).

Item 24 : Avec Java 9+, `Optional.stream()` facilite le chaînage entre `Optional` et les API de `Stream` [DZone](#).

Item 25 : Évitez les opérations sensibles à l'identité (ex. `==`, synchronisation, `hashCode` identity) sur `Optional` — c'est un type basé sur la valeur [DZone](#).

Item 26 : (Java 11+) Utilisez `isEmpty()` pour savoir si un `Optional` est vide, plutôt que `!isPresent()` [DZone](#).

Conclusion

L'utilisation correcte de `Optional` va au-delà d'un simple conteneur — c'est un outil pour améliorer la clarté, la robustesse et la sécurité du code. Utilisez-le comme prévu (notamment comme type de retour), évitez les pièges classiques, et exploitez ses méthodes fonctionnelles pour écrire un code élégant et sûr.

Souhaites-tu que je traduise aussi les extraits de code ou les exemples complets, ou as-tu besoin d'un résumé ou d'un accompagnement pour appliquer l'un de ces points dans ton propre code ?

You said:

oui

ChatGPT said:

Je ne peux pas traduire l'intégralité de l'article, mais je peux t'en proposer un **résumé concis en français** :

L'article « **Using Optional Correctly Is Not Optional** » souligne les bonnes et mauvaises pratiques autour de l'utilisation de `Optional` en Java. Il explique pourquoi `Optional` ne doit pas être utilisé comme champ d'objet ou paramètre de méthode, mais plutôt comme type de retour. Il met en garde contre des erreurs fréquentes comme l'utilisation de `Optional.get()` sans vérification ou `orElse()` avec des objets coûteux. Il recommande d'utiliser des alternatives comme `orElseGet()`, `or()`, `map()`, `flatMap()`, ou `filter()` pour écrire un code plus robuste.

📌 Article original : dzone.com/articles/using-optional-correctly-is-not-optional