NULL pour les nuls

1-Le NULL en java

NullPointerException

Les NullPointerException sont des exceptions qui surviennent lorsqu'on tente de d'utiliser une référence qui est nulle, qui ne pointe vers aucun objet. C'est-à-dire que l'on tente d'accéder à une méthode ou une variable d'un objet qui n'a pas été initialisé.

Les manières classiques d'empêcher ça

comparaison simple de l'objet dont on veut connaître sa valeur avec la valeur spéciale null

```
if (obj != null) {
   // Utiliser l'objet
} else {
   // Traiter le cas où obj est null
}
```

Utiliser try-catch pour capturer les exceptions NullPointerException

```
try{
    //utiliser l'objet
} catch(NullPointerException e){
    //traitement du cas ou l'objet est null
}
```

@NotNull, @Nullable

@NotNull : Cette annotation indique qu'une variable, un paramètre de méthode ou une valeur de retour ne doit pas être null. Ce signalement au développeur ainsi qu'à l'IDE aide à éviter les erreurs de type NullPointerException.

@Nullable : Cette annotation indique qu'une variable, un paramètre de méthode ou une valeur de retour peut être null. Elle informe le développeur qu'il doit vérifier la nullité de cette valeur avant de l'utiliser, ce qui améliore la sécurité du code en évitant les erreurs liées aux objets nuls.

2-Présentation du pattern Null Object

Principe du pattern

1. Substitution par un Objet Neutre

Le Null Object remplace une référence null par un objet spécifique qui représente une version "vide" ou "neutre" de la classe ou de l'interface. Au lieu de vérifier si un objet est null avant de l'utiliser, on utilise cet objet neutre qui assure un comportement sûr et ne causant pas de problème dans le code.

2. Encapsulation du Comportement de Nullité

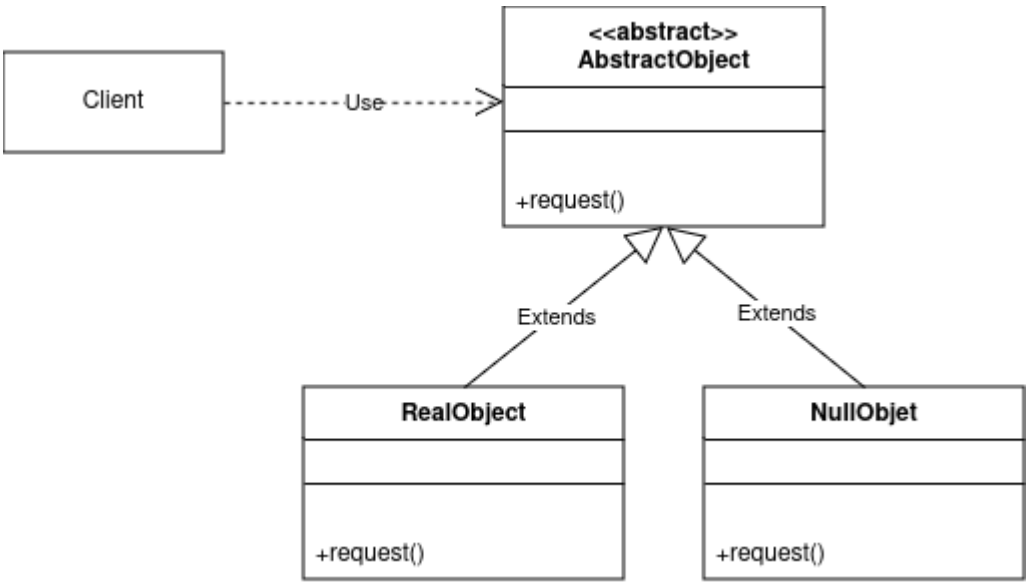
Le pattern Null Object encapsule toute la logique liée à la nullité dans une classe dédiée (le Null Object). Cela permet au code client d'utiliser cet objet sans avoir à gérer des conditions spéciales pour les références nulles, simplifiant ainsi le code et le rendant plus lisible

3. Prévention des Exceptions et Robustesse

En remplaçant null par un objet fonctionnel, le Null Object réduit le risque d'exceptions NullPointerException. Ce comportement améliore la robustesse et la stabilité de l'application en évitant les erreurs potentielles sans ajout de vérifications de nullité explicites dans le code client.

La structure du pattern:

Le pattern Null Object est constitué de plusieurs objets concrets et d'un objet null qui vont implémenter la même interface ou hériter de la même classe abstraite.



Avantages et inconvénients

Avantages

- **Lisibilité** : Le code client est plus clair car il n'a plus besoin de gérer les cas de nullité avec des structures if/else. Cela rend le code plus lisible et évite de surcharger la logique métier avec des vérifications.
- **Sécurité** : En éliminant la possibilité de références null, le pattern réduit le risque d'erreurs de type NullPointerException, renforçant ainsi la robustesse du code.
- **Maintenabilité** : En centralisant le comportement neutre dans une classe Null Object, on réduit la duplication du code de vérification et améliore la modularité. Il devient plus facile de changer le comportement "neutre" sans devoir modifier le code client.

Inconvénients

- **Complexité structurelle** : L'ajout d'une classe supplémentaire (le Null Object) pour chaque type nécessitant un comportement neutre peut rendre la structure de code plus lourde, surtout dans de grands projets où de nombreux objets utilisent ce pattern.
- **Utilité Limitée** : Le pattern Null Object est surtout efficace dans des situations où un comportement neutre est souhaitable et raisonnable. Dans les cas où la nullité représente une condition d'erreur importante, il est préférable de gérer explicitement cette situation ou de lancer une exception.

Relation avec le pattern state

La pattern state consiste à utiliser un classe abstraite pour définir plusieurs états d'un objet. On peut utiliser un NullObject en tant qu'état pour définir un état neutre/par défaut ce qui permet d'éviter la gestion avec des structures if/else d'un état null.

Relation avec le pattern strategy

Le pattern strategy consistant à définir une famille d'algorithmes, de les mettre dans des classes séparées et de rendre leurs objets interchangeables, le pattern null object s'associe parfaitement, en créant un nouvel objet implémentant l'interface mais n'ayant pas d'action (étant null).