Vers un code immuable

1) Qu'est-ce qu'un code immuable ?

Un objet immuable est une instance de classe dont les membres visibles ne peuvent être modifiés après la création. L'état de ces objets reste constant et ils sont particulièrement adaptés à la représentation de types de données abstraits comme String en Java. Une fois créés, les attributs de ces objets ne peuvent être changés.

L'immuabilité n'est pas un design pattern !

2) Intérêt de l'immuabilité

- **Thread-safe** : Les objets immuables sont sécurisés dans un environnement multi-thread car leur état ne change jamais.
- **Mise en cache** : Ils peuvent être réutilisés en mémoire, réduisant les besoins de copies.
- **Pas de constructeur par copie** : Pas besoin d'implémenter Cloneable ou des constructeurs par copie, ce qui simplifie le code.
- **Clés fiables pour Map et Set** : Les objets immuables sont stables et garantissent une récupération fiable dans les collections.

3) Création d'une classe immuable

- Déclarer la classe final pour empêcher l'héritage.
- Rendre les champs privés et finaux.
- Initialiser les champs via le constructeur.
- Ne fournir que des méthodes getter.
- Si un champ est mutable, fournir une copie défensive.

Exemple ci-dessous :

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

public final class Person {
    private final String name;
    private final int age;
    private final List<String> friends;

    public Person(String name, int age, List<String> friends) {
        this.name = name;
        this.age = age;
        // Copie défensive pour éviter que la liste mutable soit modifiée
        this.friends = new ArrayList<>(friends);
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public List<String> getFriends() {
        return new ArrayList<>(friends);
    }

    // Pas de setter, la classe est immuable
```

4) Les records et l'immuabilité

Les records introduits dans Java 14 permettent de simplifier la création d'objets immuables en générant automatiquement les méthodes getter, equals(), hashCode(), et toString(). Ils offrent une manière concise et lisible de créer des objets immuables, ce qui est utile dans des environnements multi-thread.

Conclusion

L'immuabilité permet de sécuriser les objets et les données partagées, notamment dans les environnements multi-thread, tout en rendant le code plus simple et maintenable grâce aux optimisations comme le caching et l'absence de duplication d'objets.