

Vers un c☺de immuable




Mattéo VIDOR

Martin VALET

Léo TANGUY

Sacha TROUVÉ

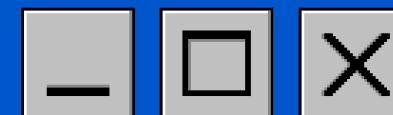
Vers un code  
**I**MMUABLE



Start



# Vers un code immuable



Mattéo VIDOR

Martin VALET

Léo TANGUY

Sacha TROUVÉ

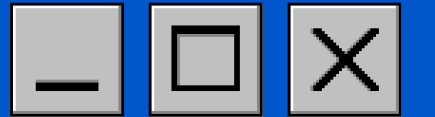


## Sommaire



- |   |                        |
|---|------------------------|
| 0 | Qu'est-ce que c'est ?  |
| 1 | Des intérêts           |
| 2 | L'immuabilité en Java; |
| 3 | Les Collections        |
| 4 | Les Records            |
| 5 | Live Coding            |
| 6 | QCM Kahoot             |
| 7 | Conclusion             |
| 8 | Bibliographie          |





Qu'est-ce que c'est ?



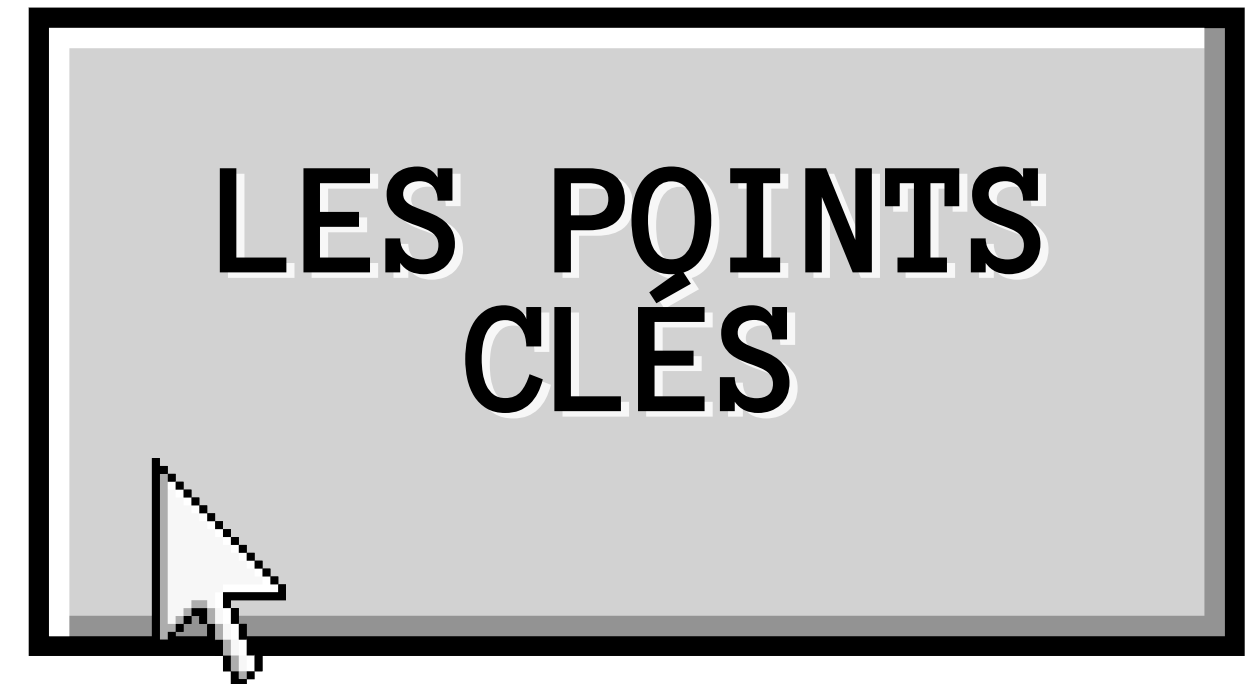
## DEFINITION

Un code `immutable` fait référence à un objet dont l'état ne peut pas être modifié.



## Les intérêts

- Lisibilité améliorée
- Les variables restent identiques
- Une meilleure traçabilité
- Une meilleure sécurité



## L'immuabilité en Java

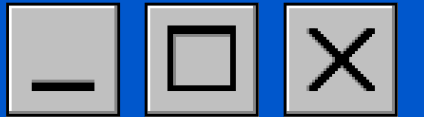
### Comment la mettre en place :

- Déclarer la classe final
- Déclarer tous les attributs comme finaux
- Fournir un unique constructeur pour initialiser tous les attributs
- Ne fournir que des "getters" et aucune méthode de modification ("setters")
- Gestion des Attributs Mutables



**LES 5 POINTS  
CLÉS**

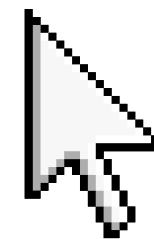




## Exemple d'un code immuable en Java

```
public final class Personne {  
    private final String nom;  
    private final String prenom;  
    private final int age;  
  
    public Personne(String nom, String prenom, int age) {  
        this.nom = nom;  
        this.prenom = prenom;  
        this.age = age;  
    }  
  
    public String getNom() {  
        return nom;  
    }  
  
    public String getPrenom() {  
        return prenom;  
    }  
  
    public int getAge() {  
        return age;  
    }  
}
```

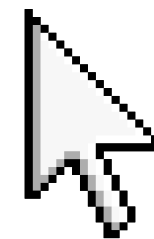
**UN CODE IMMuable**



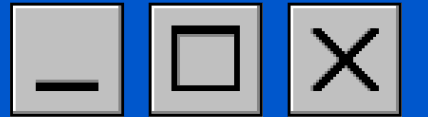
## Quid des collections

### LES COLLECTIONS

- Les Vues Non Modifiables
- Les Collections Immuables







### LES COLLECTIONS

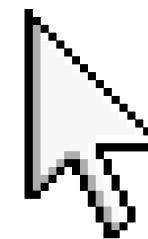
## Comment les mettre en place

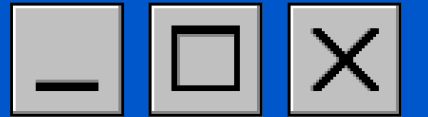
Depuis Java 9 cela est devenu très simple :

- `List.of()`, `Set.of()`, `Map.of()`

Avant Java 9 :

- Les "Wrappers" Non Modifiables





## Quid des collections(suite)

### List.copyOf()

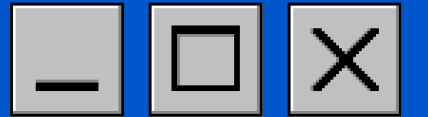
```
// Le panier du client est une liste modifiable
List<String> panier = new ArrayList<>();
panier.add("Livre");
panier.add("Stylo");

// Au moment de payer, on crée une copie immuable pour la commande
List<String> commande = List.copyOf(panier);

// Le client continue et modifie son panier après la commande
panier.add("Gomme");

// On affiche le résultat :
System.out.println("Panier final : " + panier);      // -> [Livre, Stylo, Gomme]
System.out.println("Commande passée : " + commande); // -> [Livre, Stylo]
```



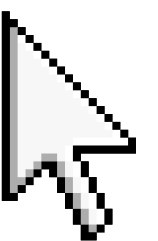


## Var

rendre le code plus  
lisible et moins verbeux

```
// Le compilateur devine que c'est un String
var prenom = "Alex";

// Le compilateur devine que c'est une Map<String, Integer>
var ages = new HashMap<String, Integer>();
ages.put("Alex", 20);
ages.put("Marie", 22);
```



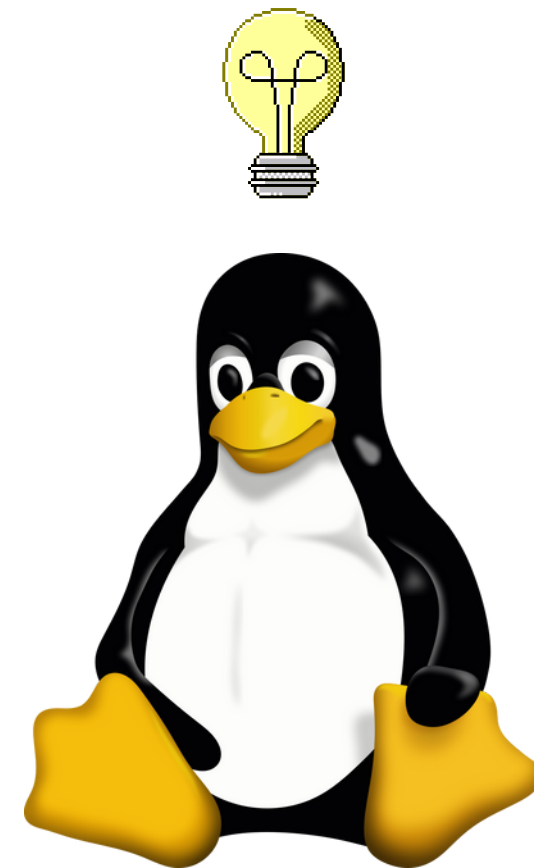


Qu'est-ce qu'un Record ?



## RECORD

- Type de classe finale
- Syntaxe spéciale à la classe



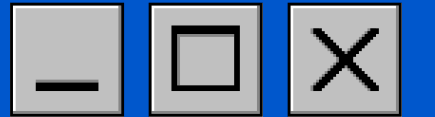
## Exemple pour la classe Personne

```
public record Personne(String nom, String prenom, int age) { }
```

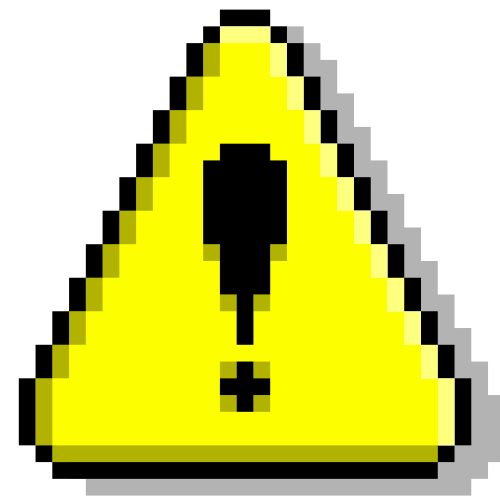


*Syntaxe légère*





## Points Notables



- Tous les éléments sont finaux
- Les getters sont en lecture seule
- Utilisation de type mutables => redéfinition de leur accès
- Pas de constructeurs, ou sinon par de redéfinition de valeur



## Utilisation d'Interfaces

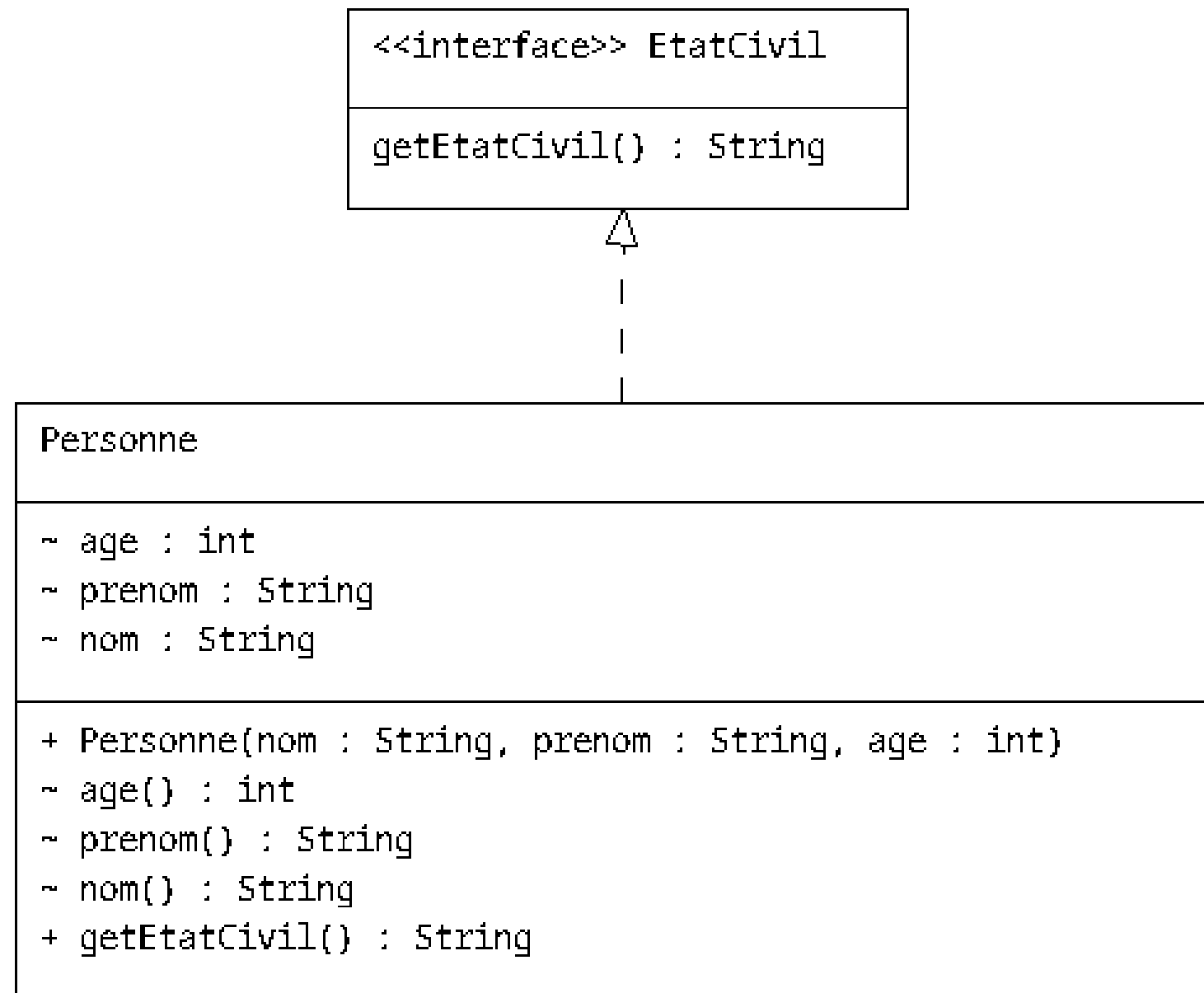
**IMPLÉMENTATION  
D'INTERFACE**



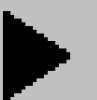
C'est possible ?



## Utilisation d'Interfaces



```
public interface EtatCivil {
    String getEtatCivil();
}
```





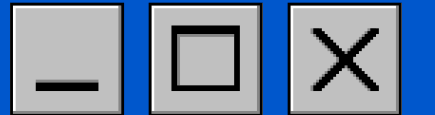
Vers un code immuable



Utilisation d'Interfaces

COMMENT ON FAIT ?





## Utilisation d'Interfaces

```
public record Personne(String nom, String prenom, int age) implements EtatCivil {  
    @Override  
    public String getEtatCivil() {  
        return prenom + " " + nom;  
    }  
}
```

- Implémentation dans définition
- Redéfinition de la méthode



## Utilisation et limites

De définir de  
nouvelles méthodes

D'implémenter des  
méthodes statiques



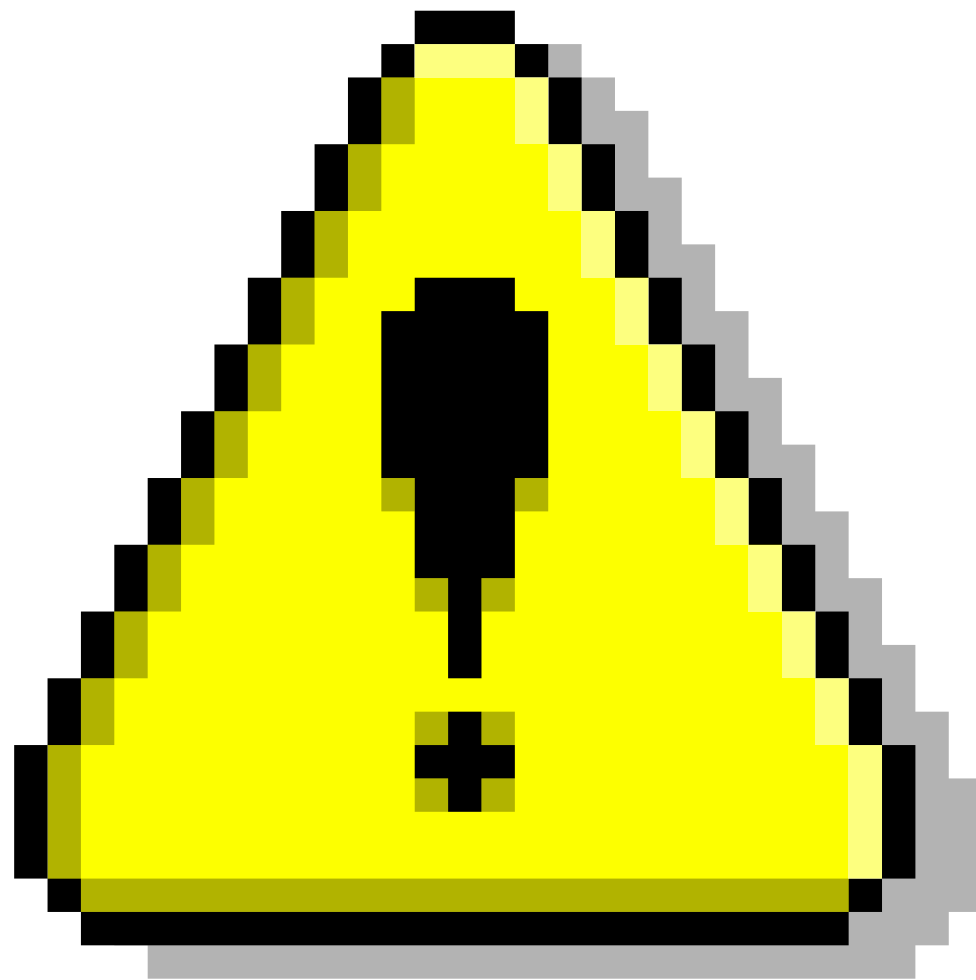
**IL RESTE  
POSSIBLE**



D'utiliser des  
types immuables, ou  
non

D'utiliser tout  
types de méthodes  
tant que ce ne sont  
pas des setters

## Utilisation et limites



**LIMITES DES  
RECORDS**



## Utilisation et limites

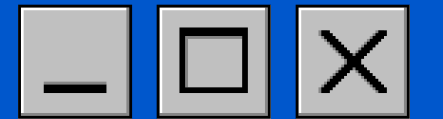
- Aucune méthodes natives



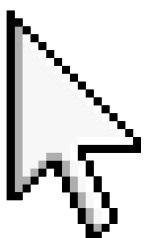
- Aucun héritage possible
- Pas de blocs d'initialisation d'instances



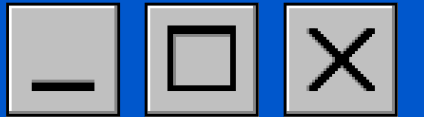
Vers un code immuable



Live coding



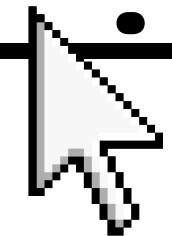
Vers un code immuable



QCM Kahoot



KAHOOT TIME !



Conclusion ;)

# L'IMMUABILITÉ

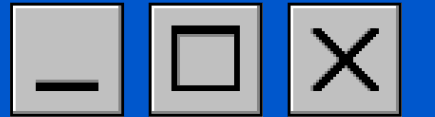
– SÛR –

– PRÉVISIBLE –

– FACILE À MAINTENIR –







## Bibliographie :

- [fr.wikipedia.org/wiki/Objet\\_immuable](https://fr.wikipedia.org/wiki/Objet_immuable)
- [devuniversity.com](https://devuniversity.com)
- [docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html)
- [docs.vavr.io](https://docs.vavr.io)

