



VERS UN CODE IMMUTABLE

BORDES Louis - DANVIN Corentin - DEDET Valentin - DRON Jeremy

Contexte

- Étudiant businessman
- Achat/Vente de positionnement
- historique des actions inexistantes

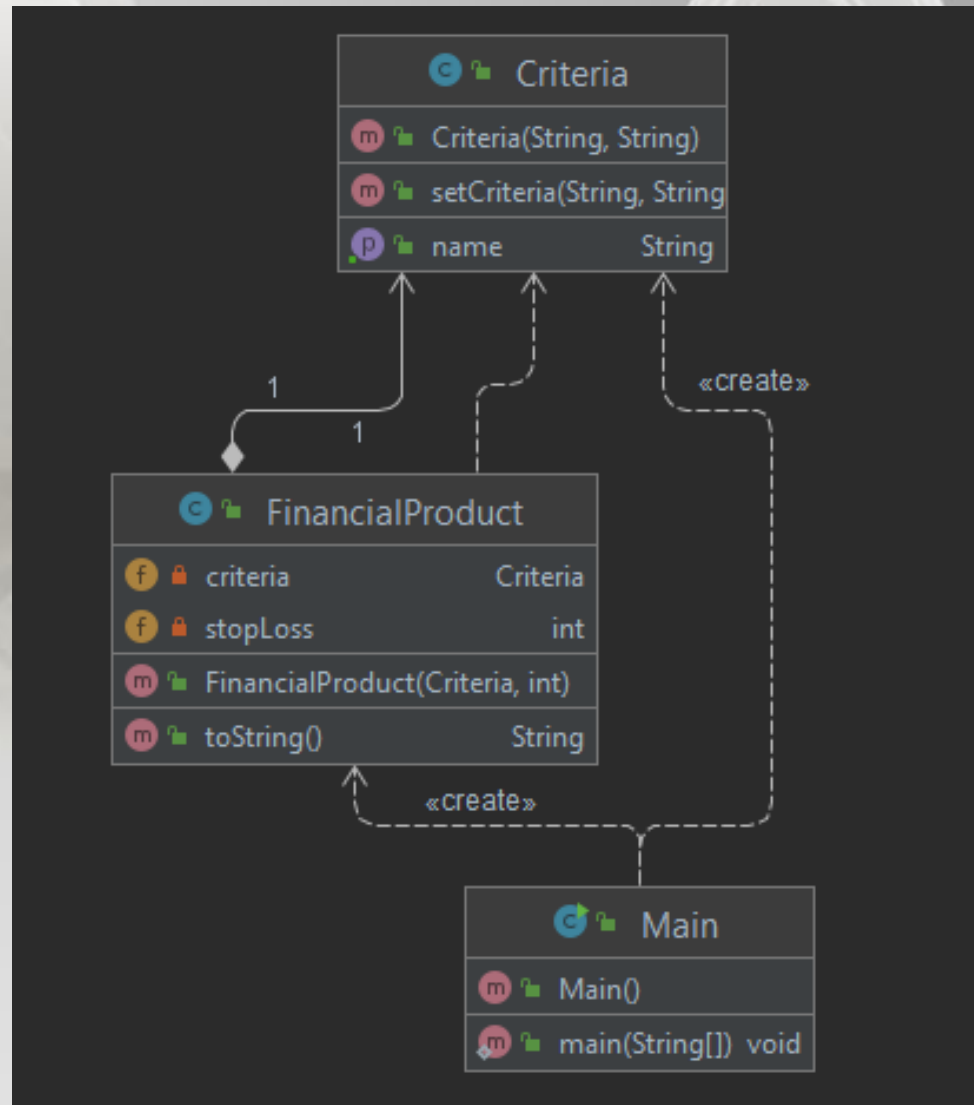


On code !

```
public class Criteria {  
    private String name;  
    private String type;  
  
    public Criteria(String name, String type) {  
        this.name = name;  
        this.type = type;  
    }  
  
    public void setCriteria(String name, String type) {  
        this.name = name;  
        this.type = type;  
    }  
  
    public String getType() {  
        return type;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

```
public class FinancialProduct{  
    private Criteria criteria;  
    private int stopLoss;  
  
    public FinancialProduct(Criteria criteria, int stopLoss) {  
        this.criteria = criteria;  
        this.stopLoss = stopLoss;  
    }  
  
    @Override  
    public String toString() {  
        return "Order Buy{" +  
            "name='" + criteria.getName() + '\'' +  
            ", type='" + criteria.getType() + '\'' +  
            ", stopLoss=" + stopLoss +  
            '\'';'  
    }  
}
```

On code !



On code !

```
public class Main {  
    public static void main(String[] args) {  
        Criteria criteria = new Criteria("Tesla", "Action");  
  
        FinancialProduct order1 = new FinancialProduct(criteria, 1200);  
        System.out.println(order1 + "\n");  
    }  
}
```

```
Order Buy{name='Tesla', type='Action', stopLoss=1200}
```

Ça fonctionne 😊

```
public class Main {  
    public static void main(String[] args) {  
        Criteria criteria = new Criteria("Tesla", "Action");  
  
        FinancialProduct order1 = new FinancialProduct(criteria, 1200);  
        System.out.println(order1 + "\n");  
  
        criteria.setCriteria("Meta", "Obligation");  
        FinancialProduct order2 = new FinancialProduct(criteria, 290);  
  
        System.out.println(order1);  
        System.out.println(order2);  
    }  
}
```

```
Order Buy{name='Meta', type='Obligation', stopLoss=1200}  
Order Buy{name='Meta', type='Obligation', stopLoss=290}
```

Ça ne fonctionne plus 😞

Résolution du problème

```
public class Main {  
    public static void main(String[] args) {  
        Criteria criteria1 = new Criteria("Tesla", "Action");  
  
        FinancialProduct order1 = new FinancialProduct(criteria1, 1200);  
        System.out.println(order1 + "\n");  
  
        Criteria criteria2 = new Criteria("Meta", "Obligation");  
        FinancialProduct order2 = new FinancialProduct(criteria2, 290);  
  
        System.out.println(order1);  
        System.out.println(order2);  
    }  
}
```

```
Order Buy{name='Tesla', type='Action', stopLoss=1200}  
Order Buy{name='Meta', type='Obligation', stopLoss=290}
```

Notre problème est résolu temporairement... si un utilisateur du programme vient à modifier les critères plutôt que de recréer une instance, il créera un bug dans le programme. Il ne faut jamais faire confiance à un utilisateur. ☹️

Résolution du problème

```
public final class Criteria {  
    private final String name;  
    private final String type;  
  
    public Criteria(String name, String type) {  
        this.name = name;  
        this.type = type;  
    }  
  
    public String getType() {  
        return type;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

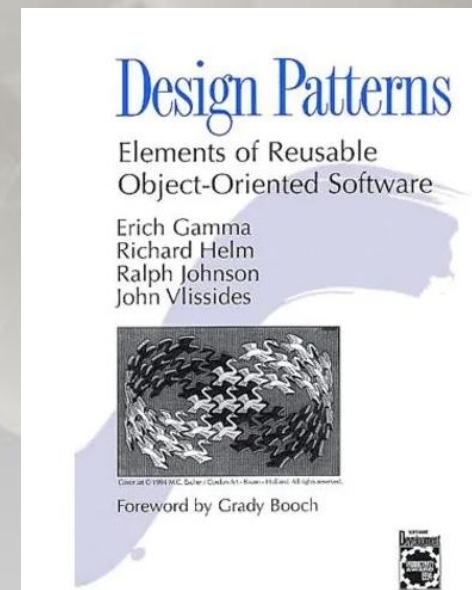
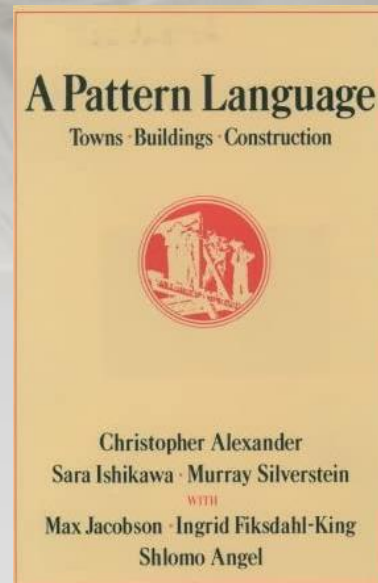
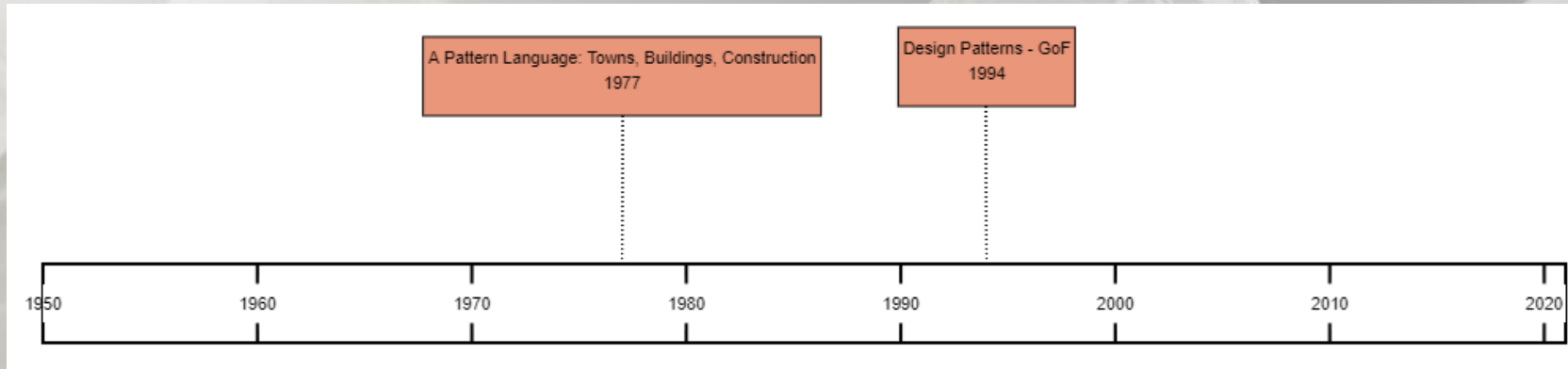
```
public record Criteria(String name, String type) {  
  
    public String getType() {  
        return type;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

Syntaxe possible depuis Java 16

```
Order Buy{name='Tesla', type='Action', stopLoss=1200}  
Order Buy{name='Meta', type='Obligation', stopLoss=290}
```

Utilisation du Design Pattern immutable. Mais qu'est-ce qu'un design pattern ? 🤔

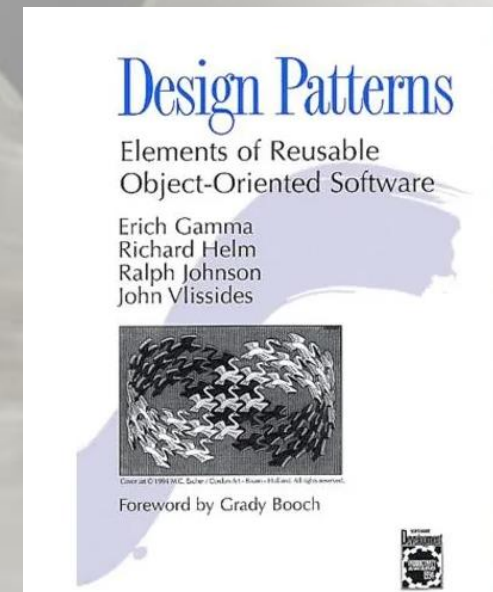
Design Pattern



Design Pattern

Classification des design patterns par le Gof:

- **Modèle de création:** Permet une optimisation de la création des objets
- **Modèle de structuration:** Permet de faire une suite de classe et d'augmenter la fonctionnalité des objets
- **Modèle de comportement:** Conçus en fonction de la façon avec laquelle les classes communiquent entre elles



Présentation du pattern

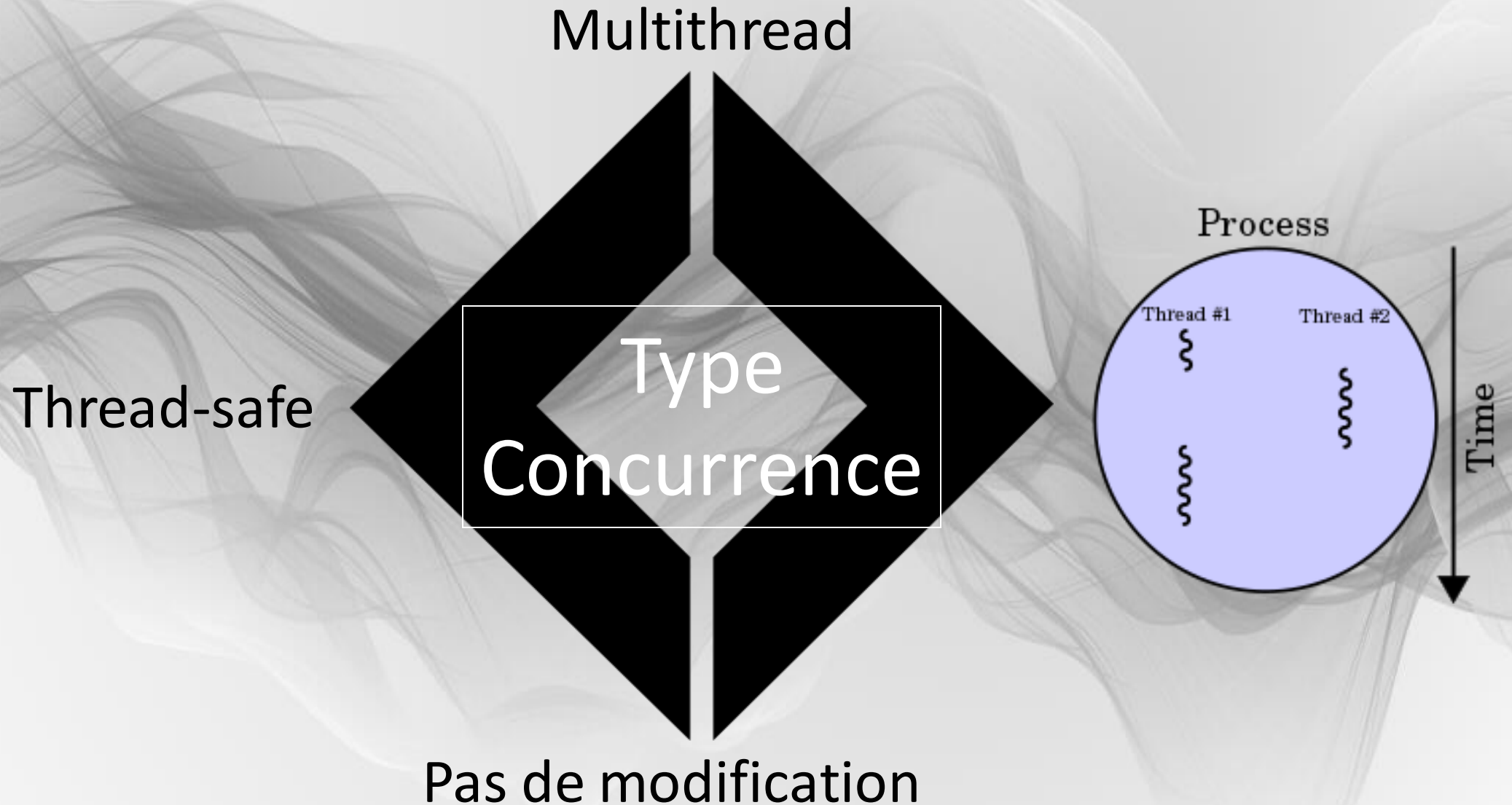
















Diagramme générique de l'immuabilité

```
public final class DateImmuable {  
  
    private final Date startDate;  
    private final Date endDate;  
  
    public DateImmuable(Date startDate, Date endDate) {  
        Date copyStart = (Date) startDate.clone();  
        Date copyEnd = (Date) endDate.clone();  
        if (copyStart.compareTo(copyEnd) > 0) {  
            throw new IllegalArgumentException("The start date is not <= the end date.");  
        }  
        this.startDate = copyStart;  
        this.endDate = copyEnd;  
    }  
  
    @Override  
    public String toString() {  
        return "The start date is " + this.startDate + " and end date is " + this.endDate ;  
    }  
  
    public Date getStartDate() { return startDate; }  
    public Date getEndDate() { return endDate; }  
}
```

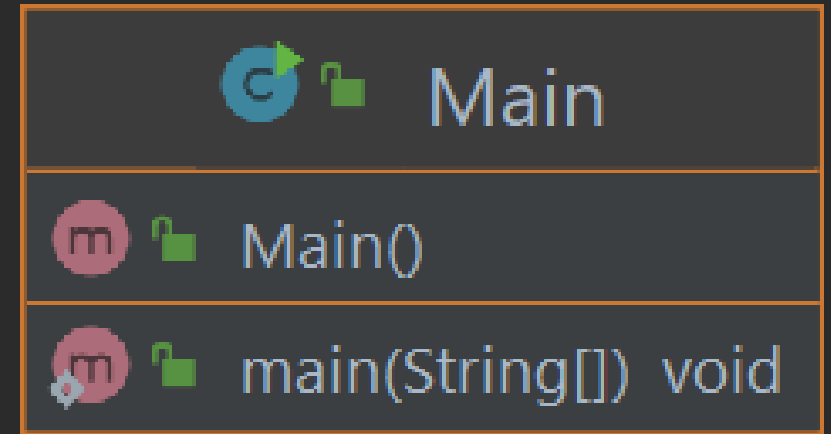
Classe
DateImmuable

		DateImmuable	
		startDate	Date
		endDate	Date
		DateImmuable(Date, Date)	
		getEndDate()	Date
		getStartDate()	Date
		toString()	String

PETIT TEST

Classe
Main

```
public class Main {  
    public static void main (String[] args) throws ParseException {  
  
        SimpleDateFormat sdf = new SimpleDateFormat();  
        sdf.applyPattern("dd/MM/yyyy");  
  
        Date dateStart = sdf.parse( source: "10/10/2021");  
        Date dateEnd = sdf.parse( source: "25/12/2021");  
  
        DateImmutable Interval1 = new DateImmutable(dateStart, dateEnd);  
  
        System.out.println(Interval1);  
    }  
}
```



The start date is Sun Oct 10 00:00:00 CEST 2021 and end date is Sat Dec 25 00:00:00 CET 2021



Mais alors ? Quels sont les liens entre l'Immuabilité et les principes SOLID ?

Les principes SOLID

- S** ➤ Single Responsibility principle = Principe de Responsabilité Unique
- O** ➤ Open-Closed Principle = Principe d'Ouverture/Fermeture
- L** ➤ Liskov Substitution Principle = Principe de Substitution de Liskov
- I** ➤ Interface Segregation Principle = Principe de Séparation des Interfaces
- D** ➤ Dependency Inversion Principle = Principe d'Inversion des Dépendances

L'Immuabilité et les principes SOLID





Les limites du pattern

Les avantages

- Thread-safe par nature
- Conseillé en environnement multi-thread
- Peut être mis en cache côté client sans risque de désynchronisation
- Peut être utilisé sans risque dans des Maps ou dans des Sets
- Quand ces objets sont utilisés comme variables d'une classe -> L'initialisation n'a pas à être fait à partir d'une copie défensive
- Plus simple et plus lisible
- Pas de Setter
- L'invariant de classe n'a besoin d'être validé qu'à la création de l'objet
- Pas nécessaire de créer un constructeur par copie

Les inconvénients

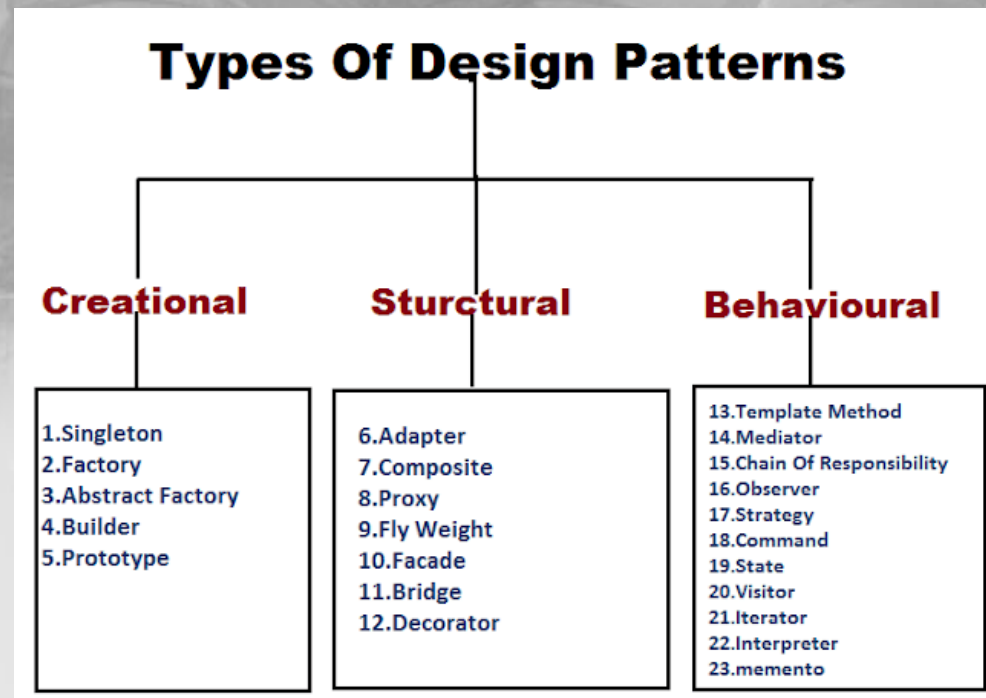
- Si on modifie un objet, il faut le recréer -> Plus coûteux qu'une simple modification
- Beaucoup de modifications -> Immuabilité faible
- Utilisation trop peu fréquente



Un pattern similaire ?

Immuabilité

- Un pattern qui n'est pas imaginé par le Gof
- Des caractéristiques similaire ?



Le pattern singleton

- Modèle de création
- Une seule instance simple
- Le constructeur de la classe est privé

Singleton

- singleton : Singleton

- Singleton()

+ getInstance() : Singleton

- Final qui assure une seule instance
- Constructeur en privé
- Instance simple

```
3 public final class Singleton {  
4  
5     private static final Singleton INSTANCE = new Singleton();  
6     private Singleton() {}  
7  
8     public static Singleton getInstance() {  
9         return INSTANCE;  
10    }  
11 }
```

Points forts

- Lisible
- Conception
- Sécurité

Points faibles

- Tests plus difficiles
- Etat trop global
- Ne respecte pas le principe de responsabilité unique

La similarité avec l'immuabilité

- Très simple et une seule instance
- On peut intégrer des instances de singleton dans une classe immuable
- Deux modèles distincts



Des classes javadoc immuables ?

Classes immuables simples

- `Java.lang.Integer`
- `Java.lang.Boolean`
- `Java.lang.Long`
- `Java.lang.Float`
- `java.lang.Character`

Autres classes

- `Java.awt.Color`
- `Java.awt.LinearGradientPaint`
- `java.net.Inet6Address`
- `java.util.Collections`

Java 16: record

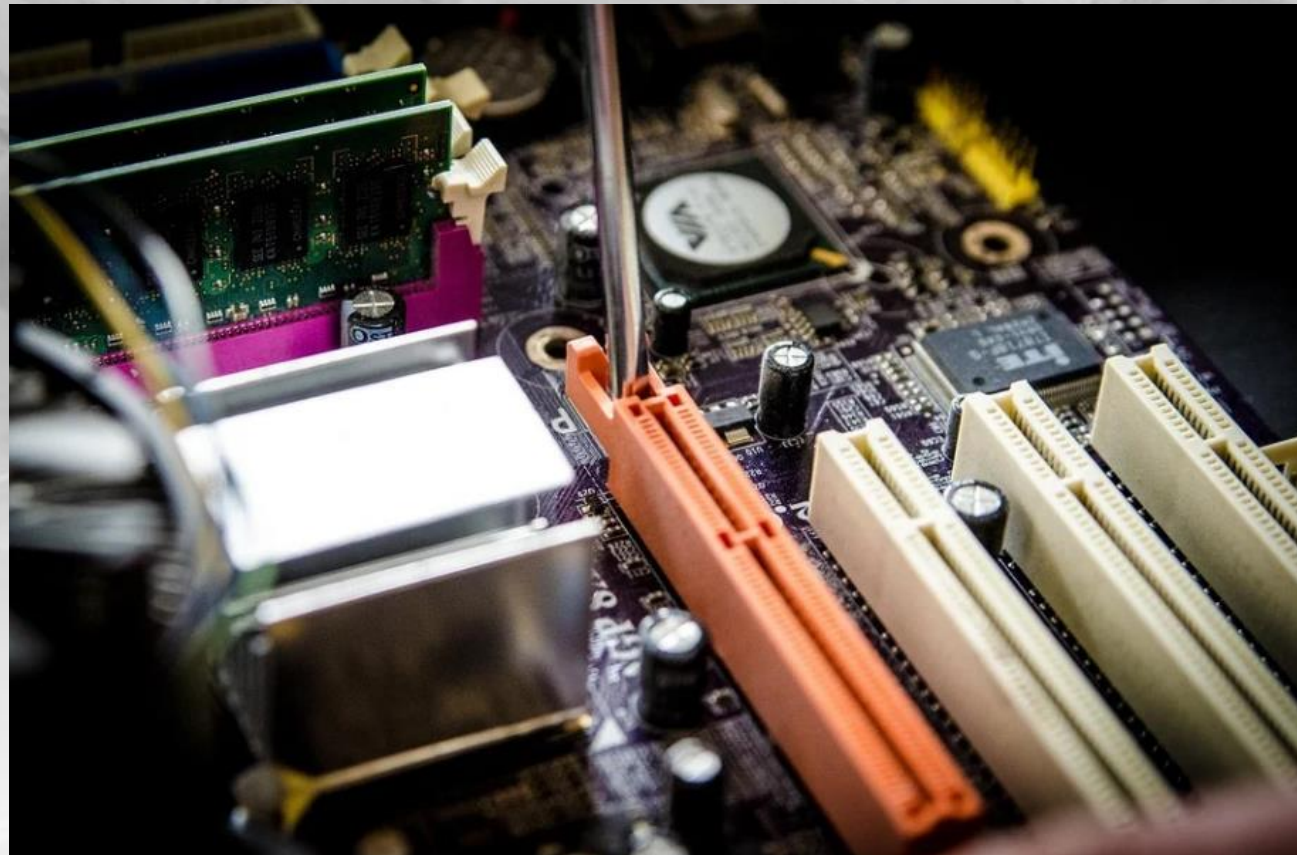
- Intégré en Java 16 (disponible en Java 14 avec l'option --enable-preview du JDK)
- Forme de classe restrictive qui rend la classe immuable
- Quelques différences

```
public final class Criteria {  
    private final String name;  
    private final String type;  
  
    public Criteria(String name, String type) {  
        this.name = name;  
        this.type = type;  
    }  
  
    public String getType() {  
        return type;  
    }  
    public String getName() {  
        return name;  
    }  
}
```

```
public record Criteria(String name, String type) {  
  
    public String getType() {  
        return type;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

Live Coding

Programme qui liste les composants d'un ordinateur 🤩



Avez-vous tout compris ? Alors... 😊

C'est l'heure du QCM !

Quelle est la définition d'une classe immuable ?

- Une classe dont les objets, une fois instanciés, ne peuvent plus changer d'état

Quel est le pattern de l'Immuabilité ?

- Concurrency

Les objets immuables sont très souvent utilisés

- Faux



Quel est le principal défaut d'un objet immuable ?

- Si l'on souhaite modifier un objet, il faut le recréer

Le pattern de l'Immuabilité est l'un des patterns du GoF ?

- Faux

Comment rendre une classe immutable ?

- La classe doit être finale
- Tous les champs doivent être déclarés final
- Ne pas créer de « setter » ou de méthode modifiant les variables de la classe
- Si l'objet contient une variable qui est une référence à une classe mutable il ne faut pas avoir de méthodes modifiant cet objet

Les classes immuables sont par nature thread-safe et elles sont
conseillées en environnement multi-thread

- Vrai

On crée une classe Date qui s'actualise tous les jours. Est-ce qu'il serait judicieux de rendre cette classe immuable ?

- Oui (même si l'option 3 est tentante)

Pour voir et revoir notre travail:

<https://github.com/Jeremod-Dev/Immutable.git>

<https://tech.io/playgrounds/59238/design-pattern>

<https://gruiz.net/quiz-answer.php?code=iSaLkT87>

Références utilisées

<https://gfx.developpez.com/tutoriel/java/immuables/#LII>

<https://wodric.com/classe-immutable/>

<https://lkumarjain.blogspot.com/2016/02/immutable-design-pattern.html>

<https://springframework.guru/gang-of-four-design-patterns/>