



Design Pattern

MVC ET SES AMIS

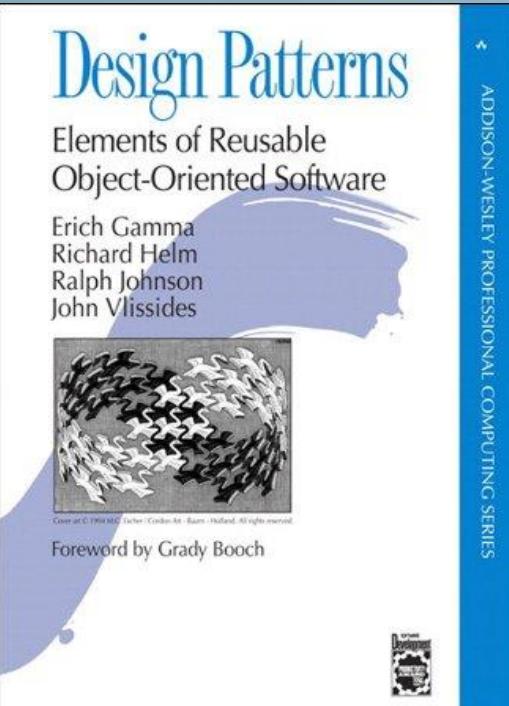
FREDERIC CANAUD – TRISTAN LEGER

GEMMA VIDAL – ANNE-SOPHIE LAVILLE

SOMMAIRE

- Qu'est-ce qu'un pattern ?
- Présentation du pattern MVC
- Exemple de système (calculatrice)
- Les Principes SOLID avec le MVC
- Présentation des petits amis du MVC : MVP et MVVM
- QCM

LES PATTERNS DE CONCEPTION



GoF - 1994

- Patterns Structuraux : Adapter, Composite, Decorator...
- Patterns Créateurs : Factory & Abstract Factory, Builder...
- Patterns Comportementaux : Strategy, Visitor, State...

Tous ces patrons de conception ont un même objectif :
Résoudre un problème récurrent par la meilleure solution possible.

LES PATTERNS D'ARCHITECTURE

- Ils concernent la structure générale du code
- Peuvent comporter des Patrons de conception
- Concernent l'ensemble du logiciel et non une ou plusieurs classes
- Exemple de pattern d'architecture : MVC, Hexagonale...

L'objectif des patterns d'architecture est :
d'apporter une solution à grande échelle sur la forme du logiciel sans se préoccuper
des détails qui, eux, seront gérés par des bonnes pratiques ou des règles.

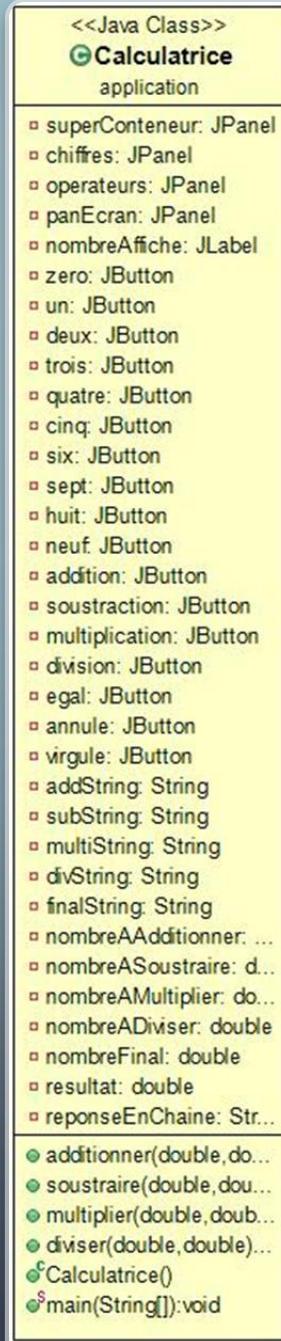
EXEMPLE DE LA PRÉSENTATION : UNE CALCULATRICE

Fonctionnalités principales de la calculatrice :

- Faire des calculs basiques (addition, soustraction, multiplication, division)
- Etre capable de réinitialiser le calcul en cours (Bouton C)
- Être capable d'enregistrer le dernier résultat afin d'effectuer un nouveau calcul sur lui.

PREMIÈRE VERSION DE NOTRE CALCULATRICE

- Beaucoup de données (5 méthodes, 17 écouteurs d'actions...) dans une seule classe
- Absence de bonnes pratiques
- Mauvaise lisibilité et organisation





COMMENT SÉPARER LES DONNÉES DE L'INTERFACE ?

ORIGINE DU PATTERN MVC

- 1978 : MVC est créé par Trygve Reenskaug, un informaticien norvégien
- Utiliser pour créer des interfaces graphiques
- Architecture de Framework



PRÉSENTATION DU PATTERN MVC

Model

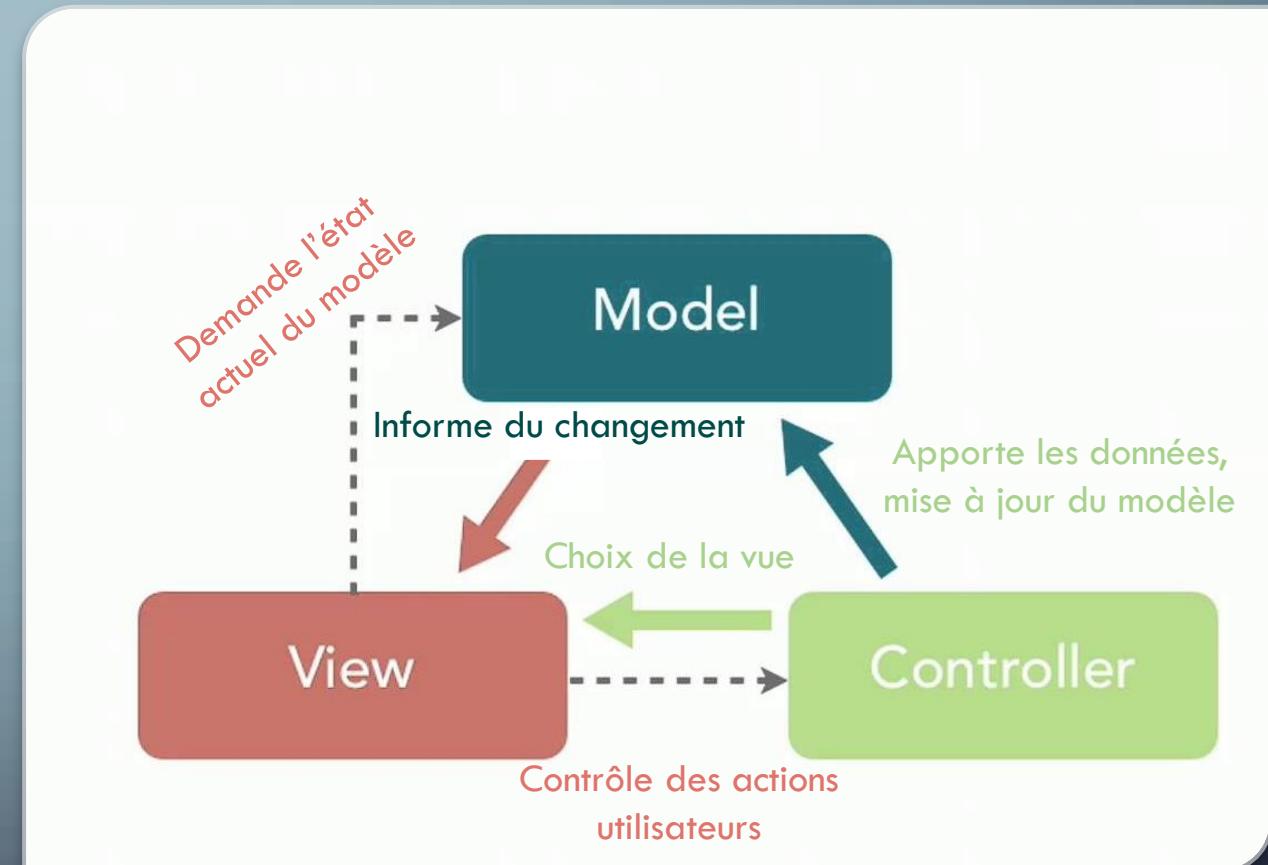
- Définit les données utilisées du code

View

- Représente l'interface graphique utilisateur

Controller

- Logique métier : traitement des requêtes
- Gère les actions de l'utilisateur et traite les données du modèle
- S'occupe de mettre à jour la vue et les données



QUELS AUTRES PATTERNS PEUT-ON RETROUVER ?

Observer

Synchronisation entre la Vue et le Modèle :

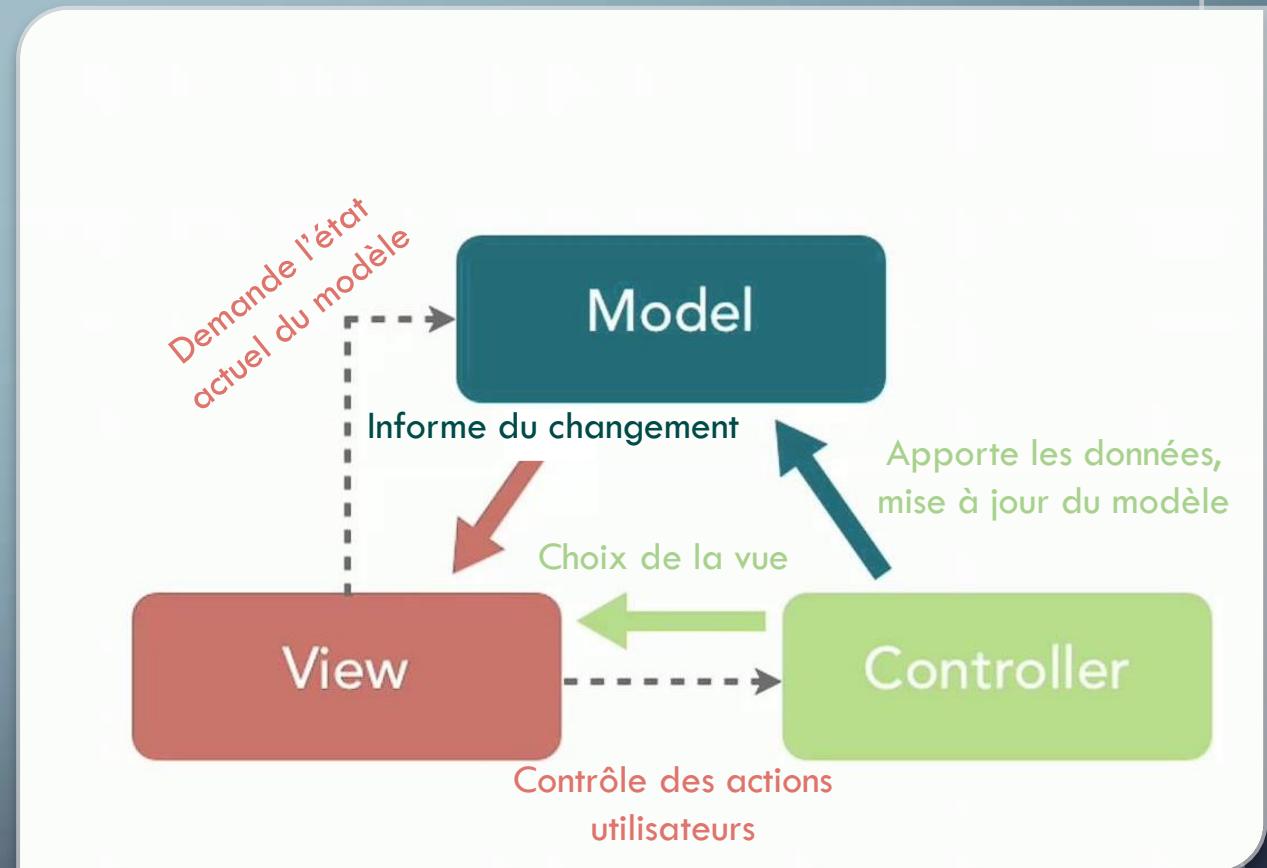
- Un observateur qui implemente la vue
- Un observable qui implémente le modèle

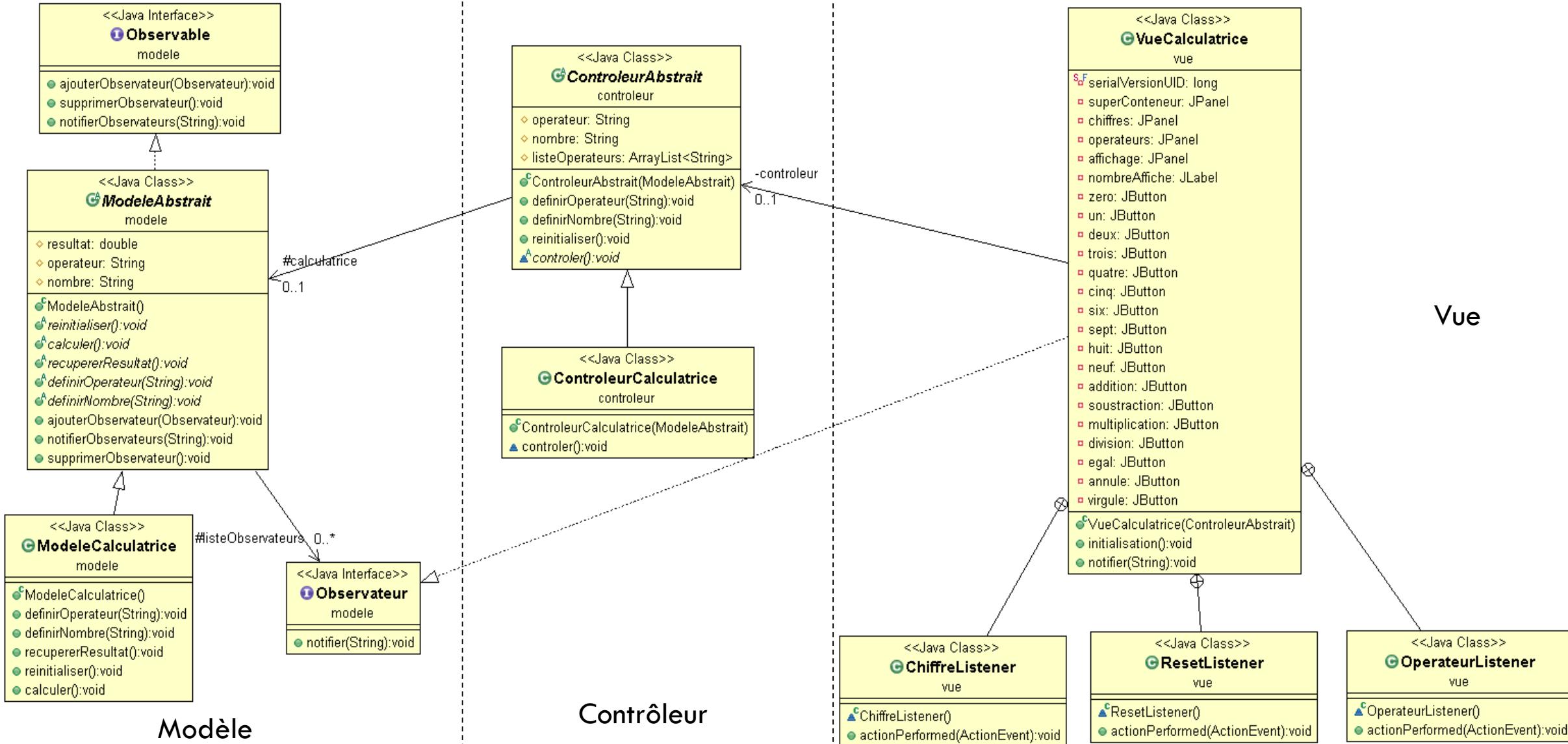
Composite

- Représentation des composants

Strategy

- Le contrôleur est la stratégie de la vue





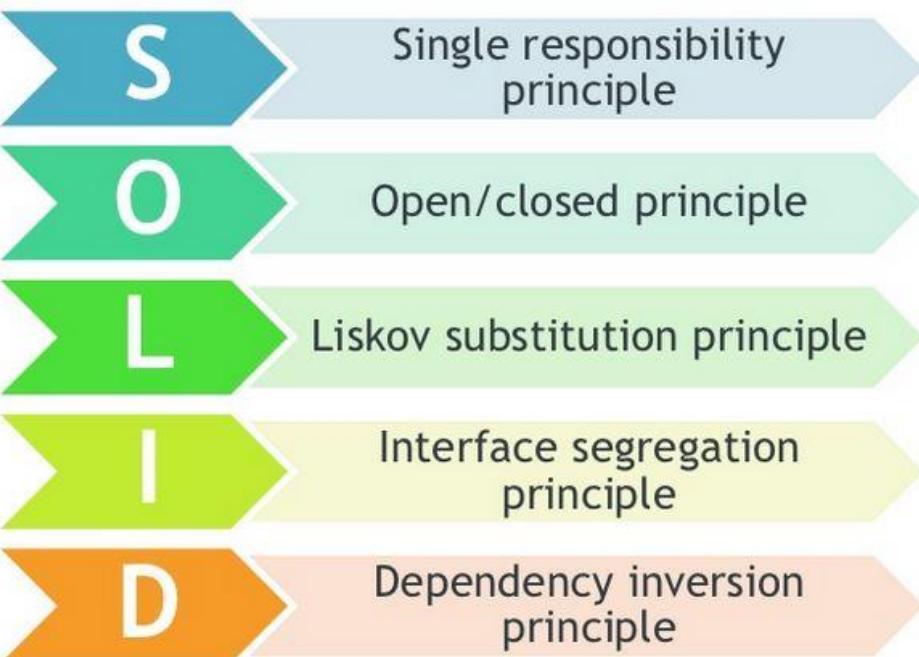
Modèle

Contrôleur

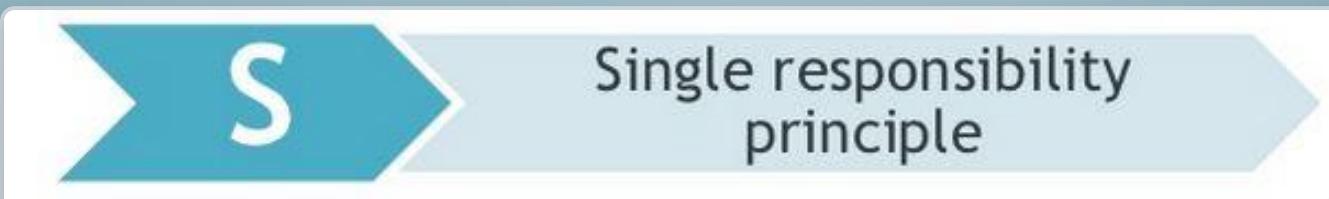
Vue

NOUVELLE VERSION DE NOTRE CALCULATRICE

LES PRINCIPES SOLID



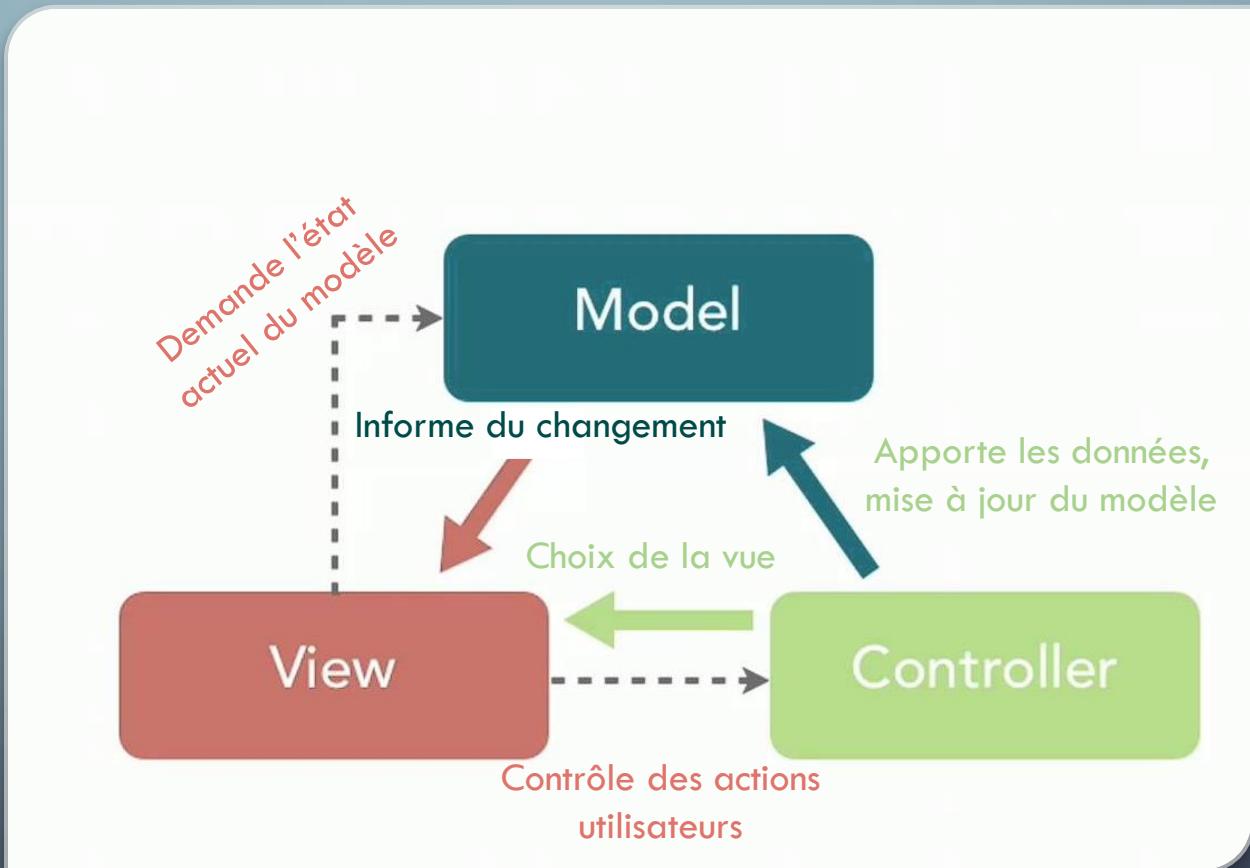
SRP (PRINCIPE DE RESPONSABILITÉ UNIQUE)



Une classe doit avoir **une et une
seule responsabilité**.

Autre définition : Une classe ne doit
avoir **qu'une seule raison de
changer**.

RESPECT DU SRP : A PREMIÈRE VUE



RESPECT DU SRP : A PREMIÈRE VUE

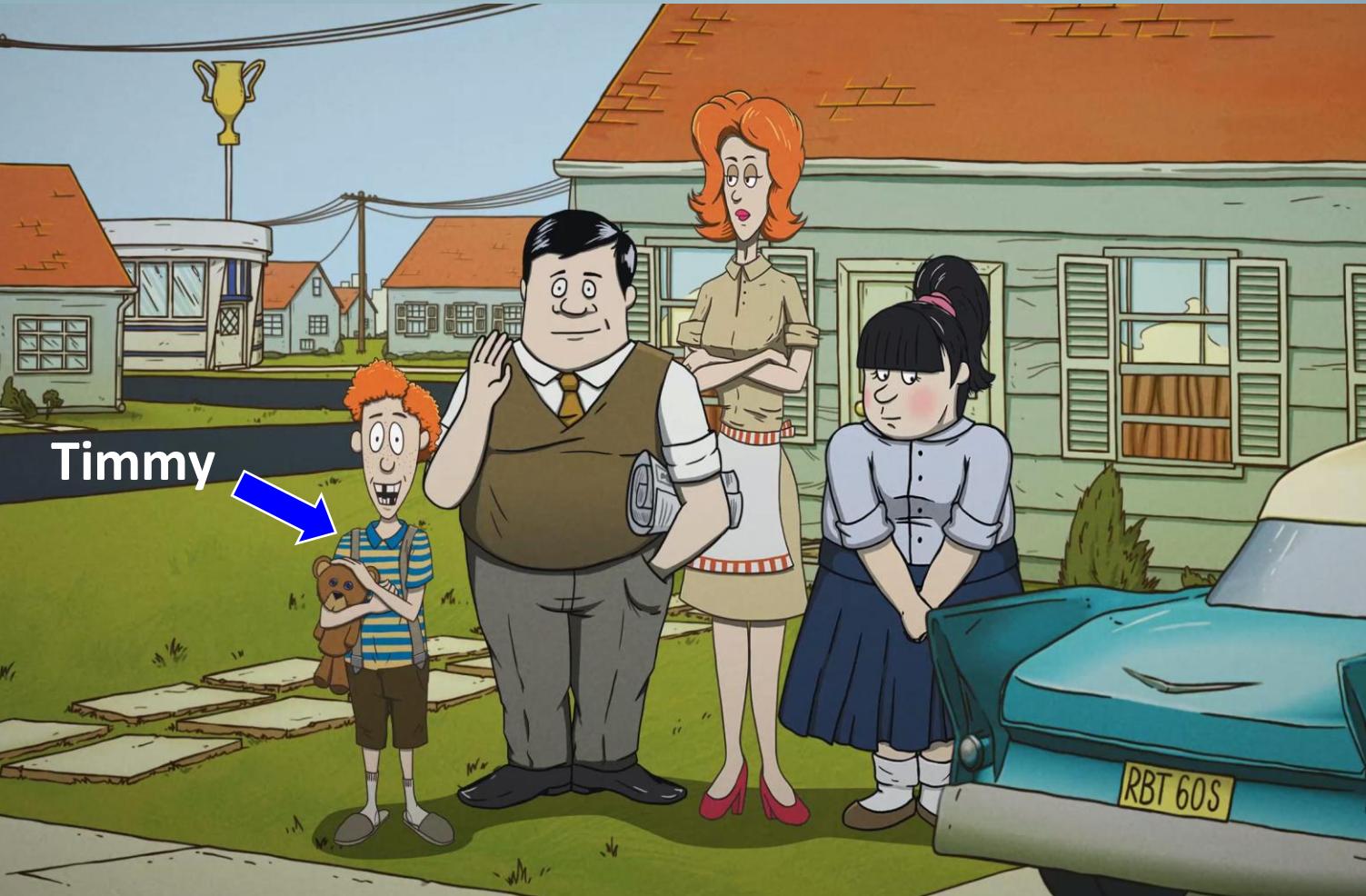
Le contrôleur doit :

- **Connaître l'interface graphique**, pour pouvoir signaler les actions de l'utilisateur
- **Connaître la logique du modèle**, pour pouvoir invoquer les méthodes appropriées sur le modèle

→ Le contrôleur a donc **2 raisons de changer**

✗ Non-respect du principe SRP ?

RESPECT DU SRP : EXEMPLE DU PETIT DÉJEUNER



Source : steamcommunity.com/games/1012880

RESPECT DU SRP : EXEMPLE DU PETIT DÉJEUNER

Chaque dimanche, Timmy doit faire le petit déjeuner, c'est à dire:

- Faire bouillir les oeufs
- Faire griller le pain
- Faire des bonnes tasses de thé vert (oui, le thé vert est le meilleur thé)

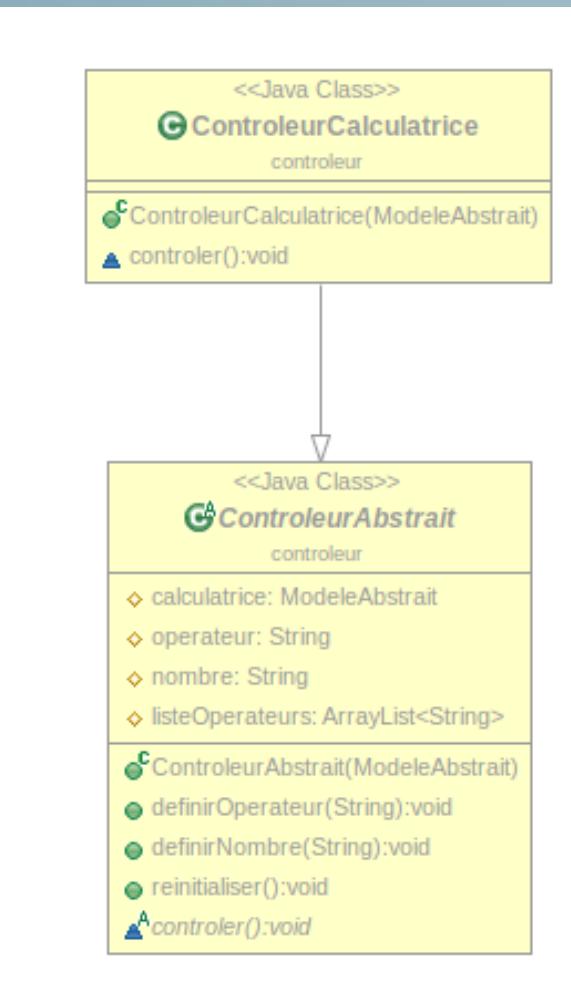
De cette façon, on peut diviser « Faire le petit-déjeuner » en ces trois petites tâches qui seront ensemble abstraites de « Faire le petit déjeuner » !

POUR RESPECTER SRP

Tout est une question de **couches d'abstraction**

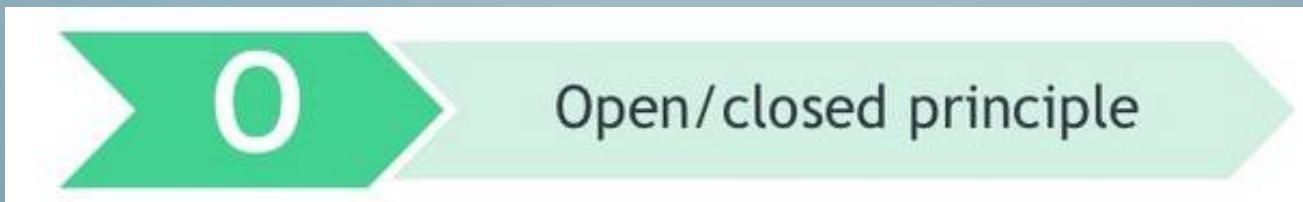
- Le contrôleur doit avoir à sa première couche une seule responsabilité
- Le contrôleur est ensuite libre de contrôler un certain nombre de tâches, si elles sont situées sur des couches inférieures d'abstraction
- Le contrôleur ne doit pas en savoir trop sur le fonctionnement des tâches, dans la mesure où il sait qu'il doit appeler les méthodes dont il dispose.

QUANT À LA CALCULATRICE...



✓ Respect du SRP !

OCP (PRINCIPE D'OUVERTURE/FERMETURE DU CODE)



Une classe doit :

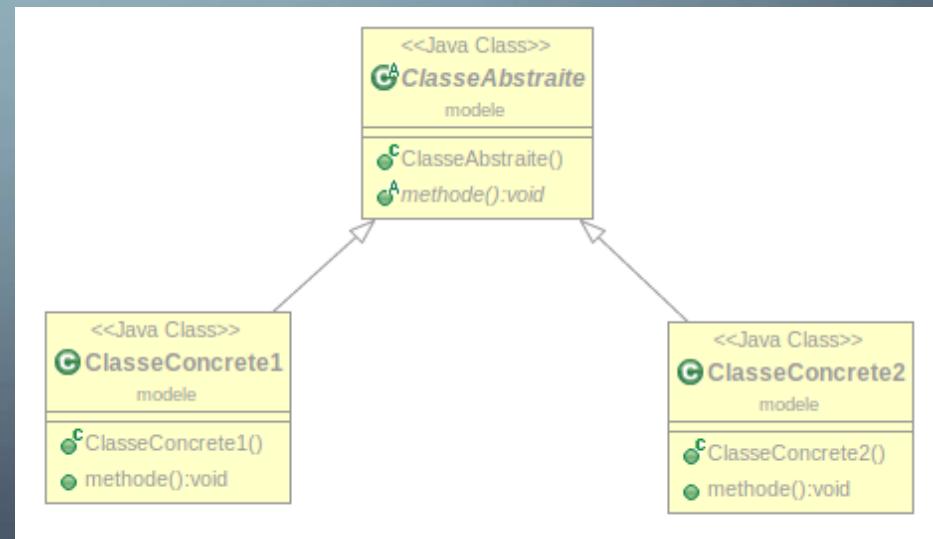
- Être **ouverte aux extensions**
- Mais **fermée aux modifications**

COMMENT RESPECTER OCP ?

- Est dû à l'ajout de fonctionnalités par modification du code existant
- Entraine l'enchaînement de conditions → BAD SMELL
- Solution : passer par l'abstraction d'une classe abstraite ou d'une interface.



Respect du OCP !



... MAIS CELA N'EST PAS TOUJOURS POSSIBLE POUR MVC !

```
@Override
public void calculer() {

    if (this.operateur.equals(Constante.CHAINNE_VIDE)) {
        this.resultat = Double.parseDouble(this.nombre);
    }

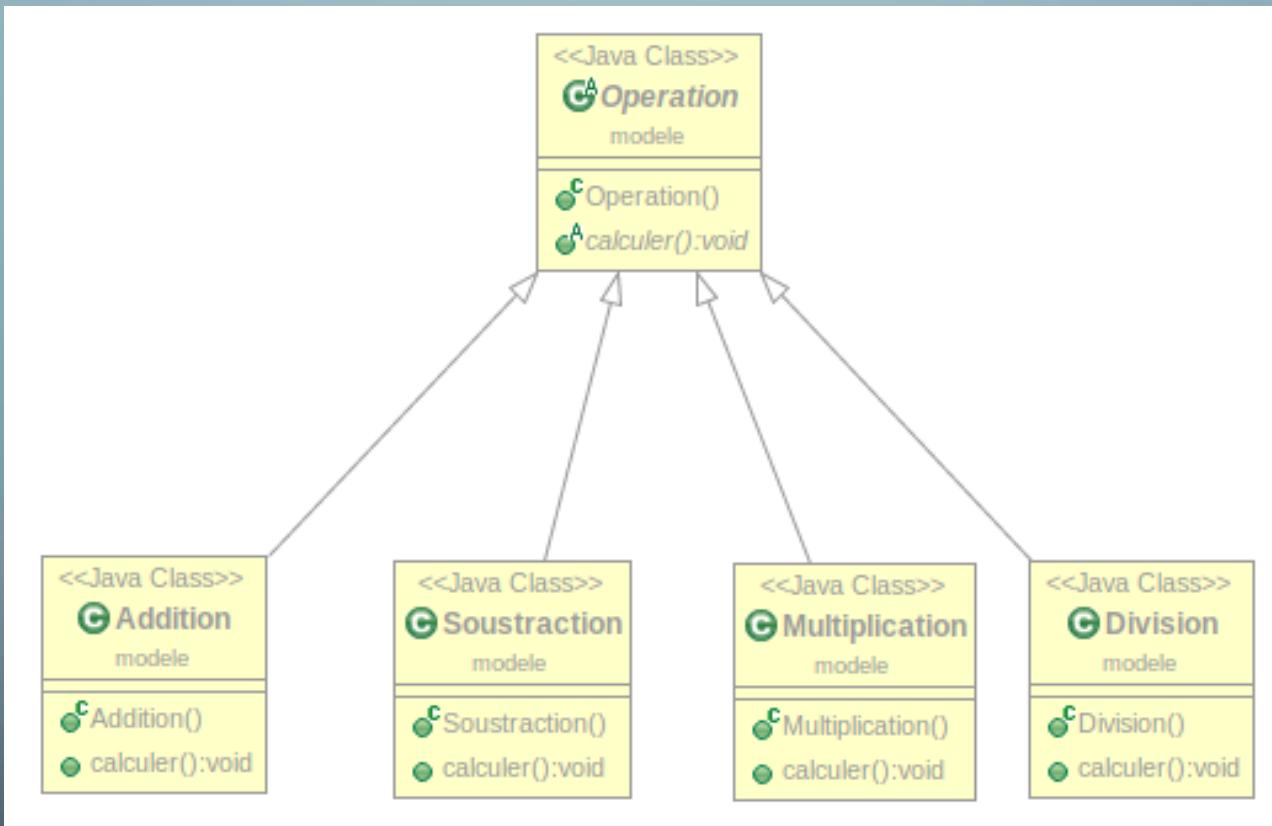
    else {
        if (!this.nombre.equals(Constante.CHAINNE_VIDE)) {
            if (this.operateur.equals(Constante.OPERATEUR_ADDITION))
                this.resultat += Double.parseDouble(this.nombre);

            else if (this.operateur.equals(Constante.OPERATEUR_SOUSTRACTION))
                this.resultat -= Double.parseDouble(this.nombre);

            else if (this.operateur.equals(Constante.OPERATEUR_MULTIPLICATION))
                this.resultat = this.resultat*Double.parseDouble(this.nombre);

            else if (this.operateur.equals(Constante.OPERATEUR_DIVISION)) {
                try {
                    this.resultat /= Double.parseDouble(this.nombre);
                } catch (ArithmeticsException e) {
                    this.resultat = 0;
                }
            }
        }
    }
}
```

UNE SOLUTION ENVISAGEABLE...

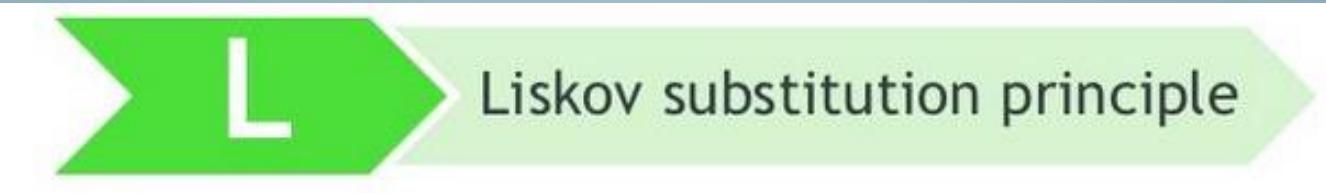


... MAIS NON RÉALISABLE !

- Or, lorsqu'on appuie sur un bouton de la calculatrice, on récupère un caractère (converti en String par la méthode `getText()`)
- D'un String à un objet de type Operation : nécessité d'une comparaison → Nécessité d'une condition de comparaison, et donc de passer par une structure IF ... OU SWITCH ...

✗ Non-respect du principe OCP

LSP (PRINCIPE DE SUBSTITUTION DE LISKOV)

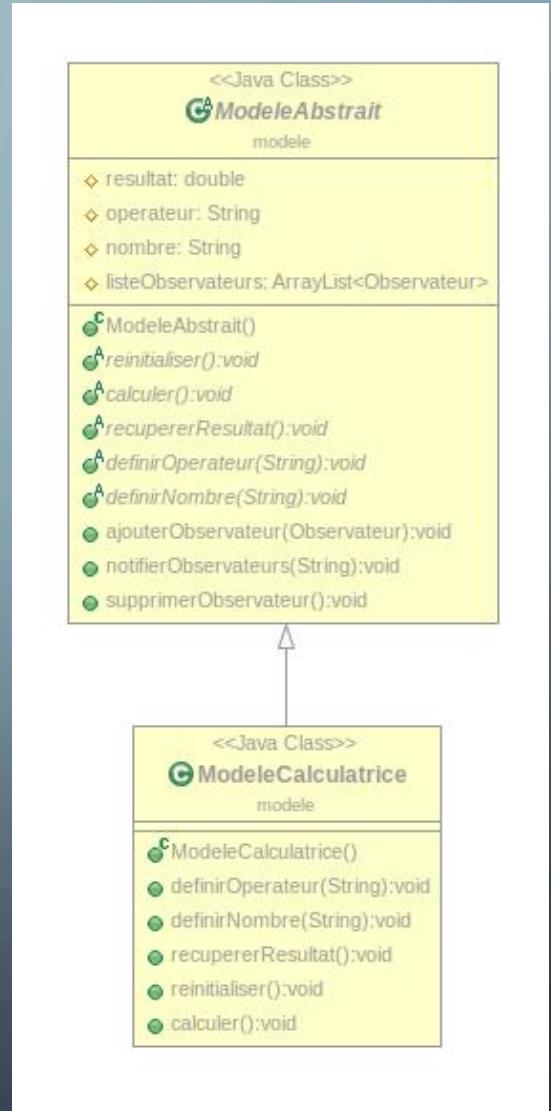
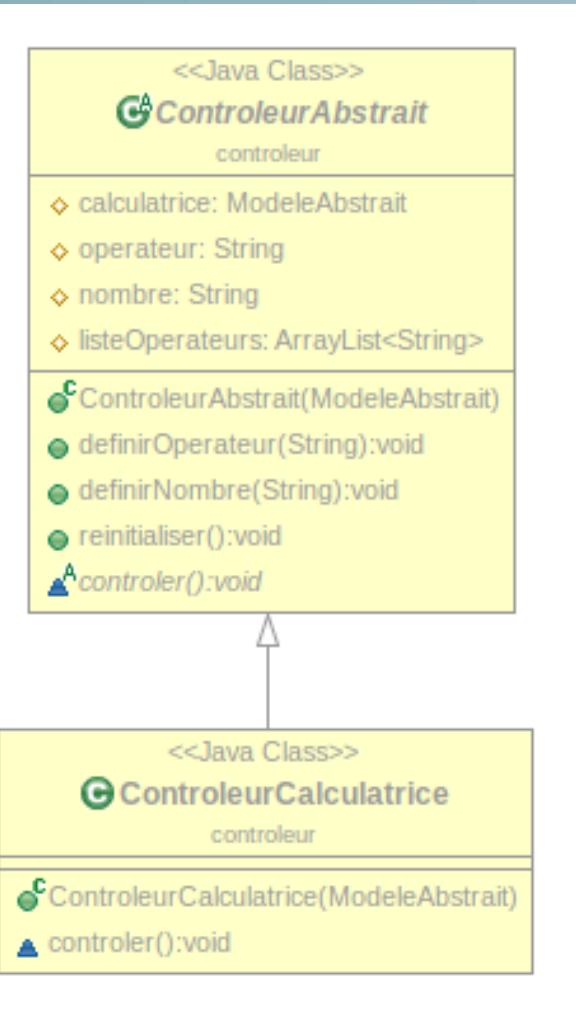


Les sous-types doivent pouvoir être
substitués à leur type de base.



Source : <https://pixabay.com/fr/photos/canard-couvert-teal-canard-4062460/>

RESPECT DU LSP



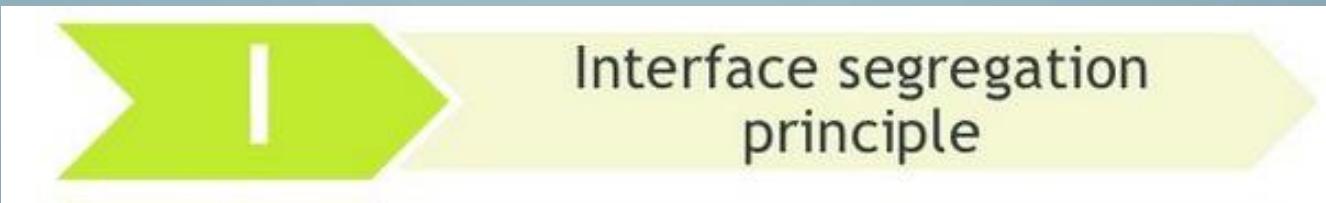
RESPECT DU LSP

- Comme les classes mères sont abstraites, on ne peut les instancier
- Le problème de nécessité d'héritage ne se pose donc pas, puisque le comportement du modèle et du contrôleur de la calculatrice sont propres à eux-mêmes.

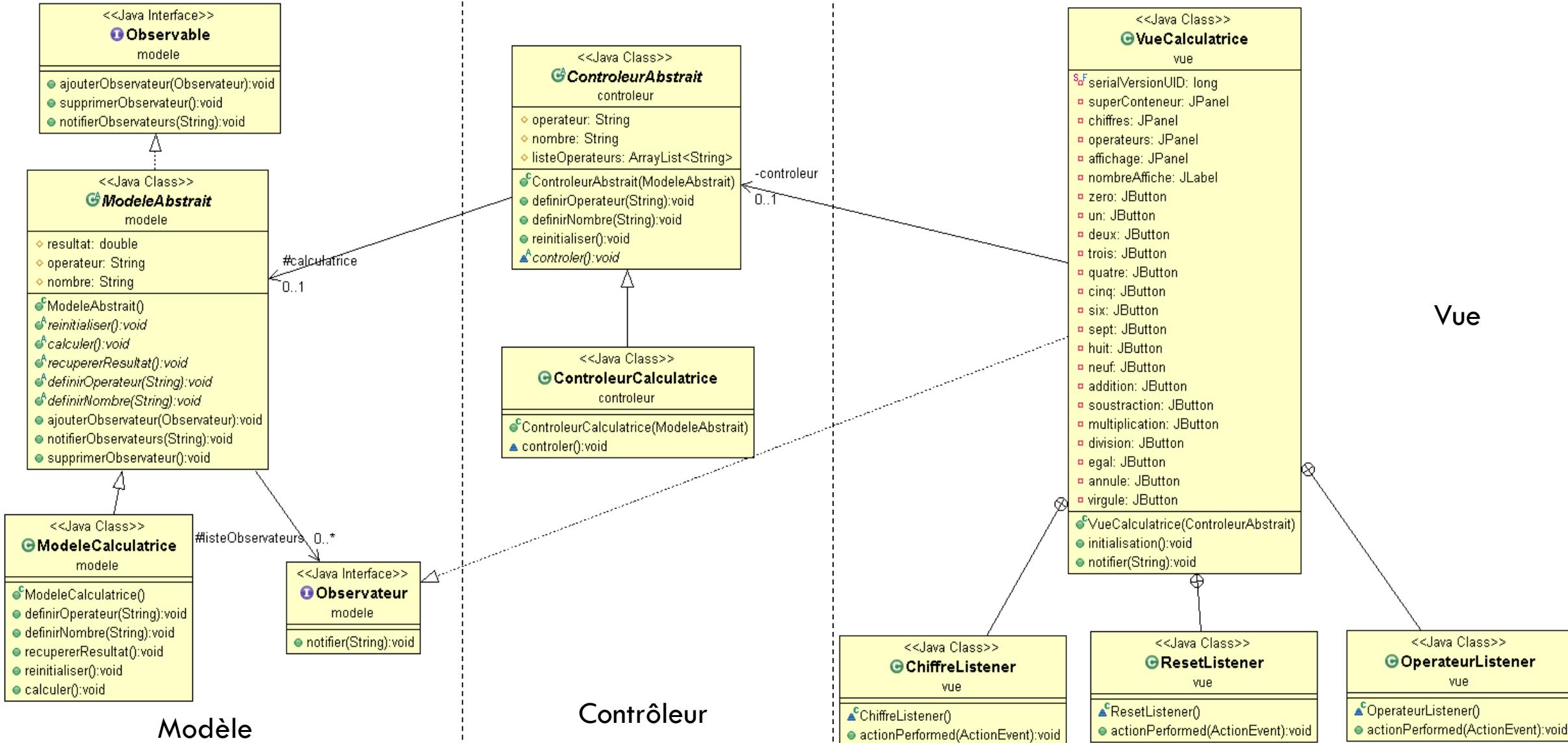


Respect du LSP !

ISP (PRINCIPE DE SÉGRÉGATION DES INTERFACES)



Une classe ne **doit pas avoir accès à des fonctions qui ne la concerne pas.**



RAPPEL DU DIAGRAMME DE CLASSE

RESPECT DU ISP

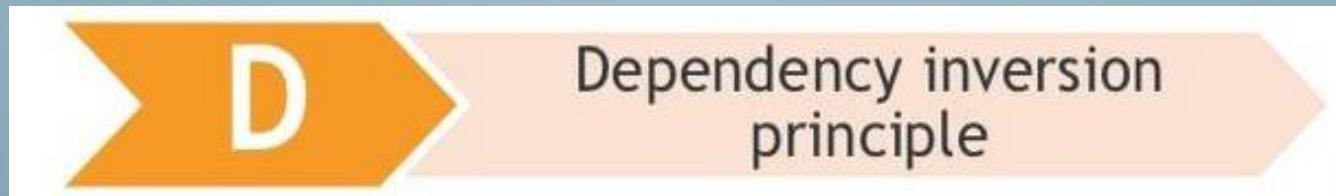
Avec le pattern Observer :

- La Vue implémente la méthode de notification de l'interface Observateur
- Le Modèle implémente les méthodes de l'interface Observable



Respect du ISP !

DIP (PRINCIPE D'INVERSION DES DÉPENDANCES)



Les modules de **haut niveau** doivent
pas **dépendre** des modules de **plus bas
niveau**.

Il faut dépendre des **abstractions**, pas
des **implémentations**

RESPECT DU DIP

- Le modèle concret dépend d'un modèle abstrait
- Le contrôleur concret dépend d'un contrôleur abstrait
- De ce fait, le couplage entre l'abstraction et le concret étant faible, l'ajout d'un deuxième contrôleur ou modèle concret de calculatrice se fait aisément !

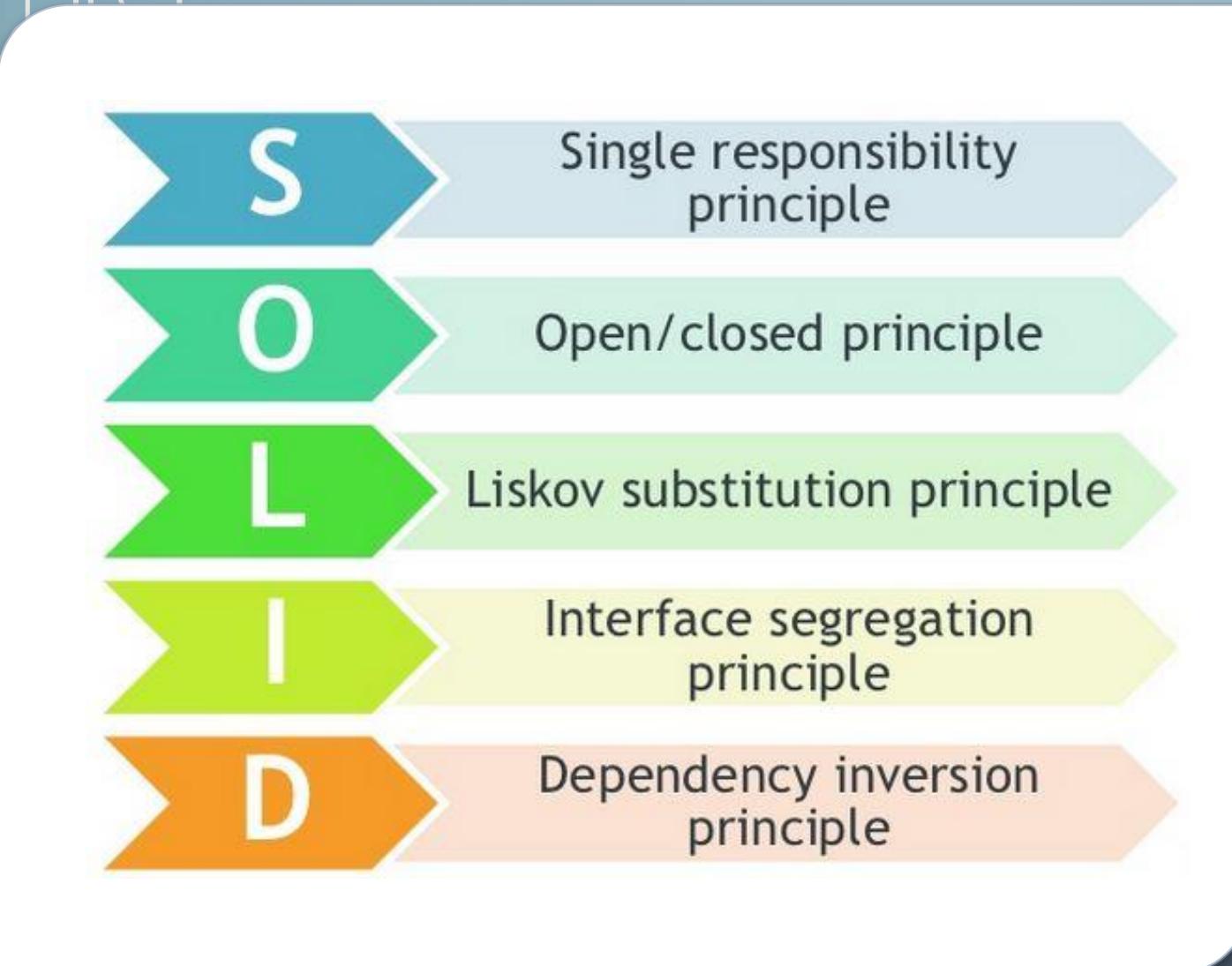


Respect du DIP !

POUR BIEN CODER DES APPLICATIONS AVEC LE PATTERN MVC

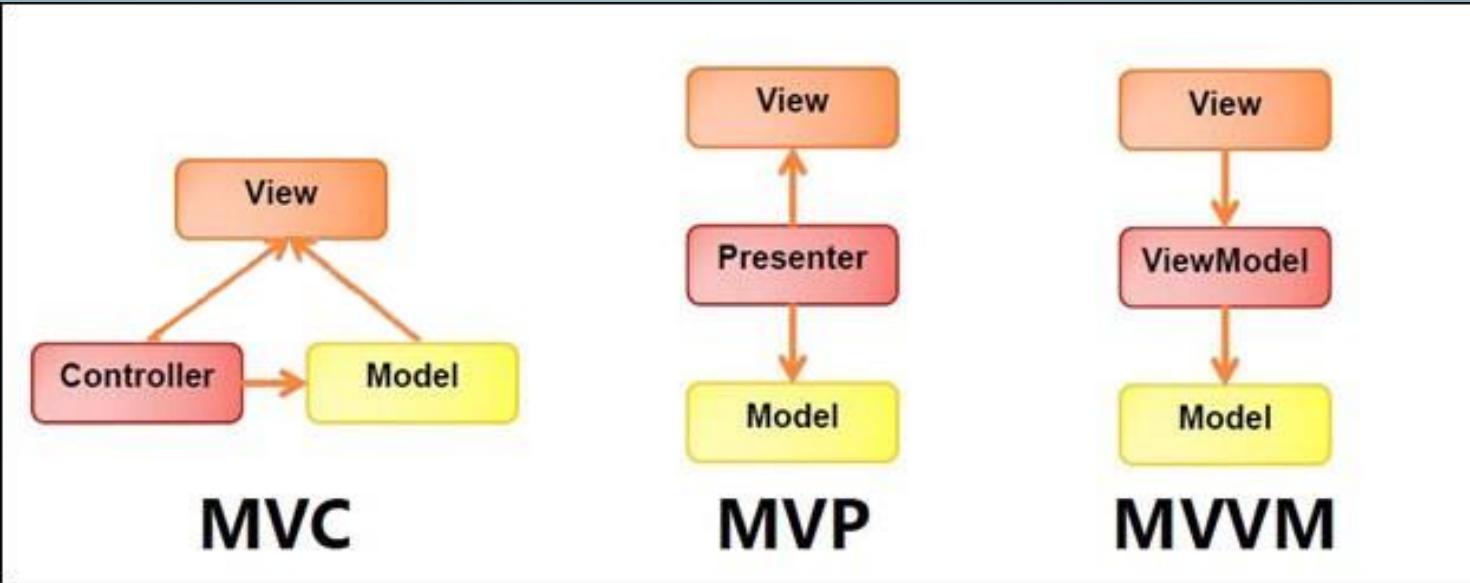
- Utiliser des interfaces suffisamment « granulées », avec uniquement des fonctionnalités communes
- Utiliser des classes abstraites autant que possible, pour encapsuler les fonctionnalités communes
- Minimiser la dépendance vis-à-vis des entités externes. En découplant l'architecture via des classes abstraites, on minimise le risque de casser toute l'application avec le changement d'une entité !

RESPECT DES PRINCIPES SOLID



Pas toujours respecté, cela dépend des types de données échangés entre les modules

LES PETITS AMIS DU MVC

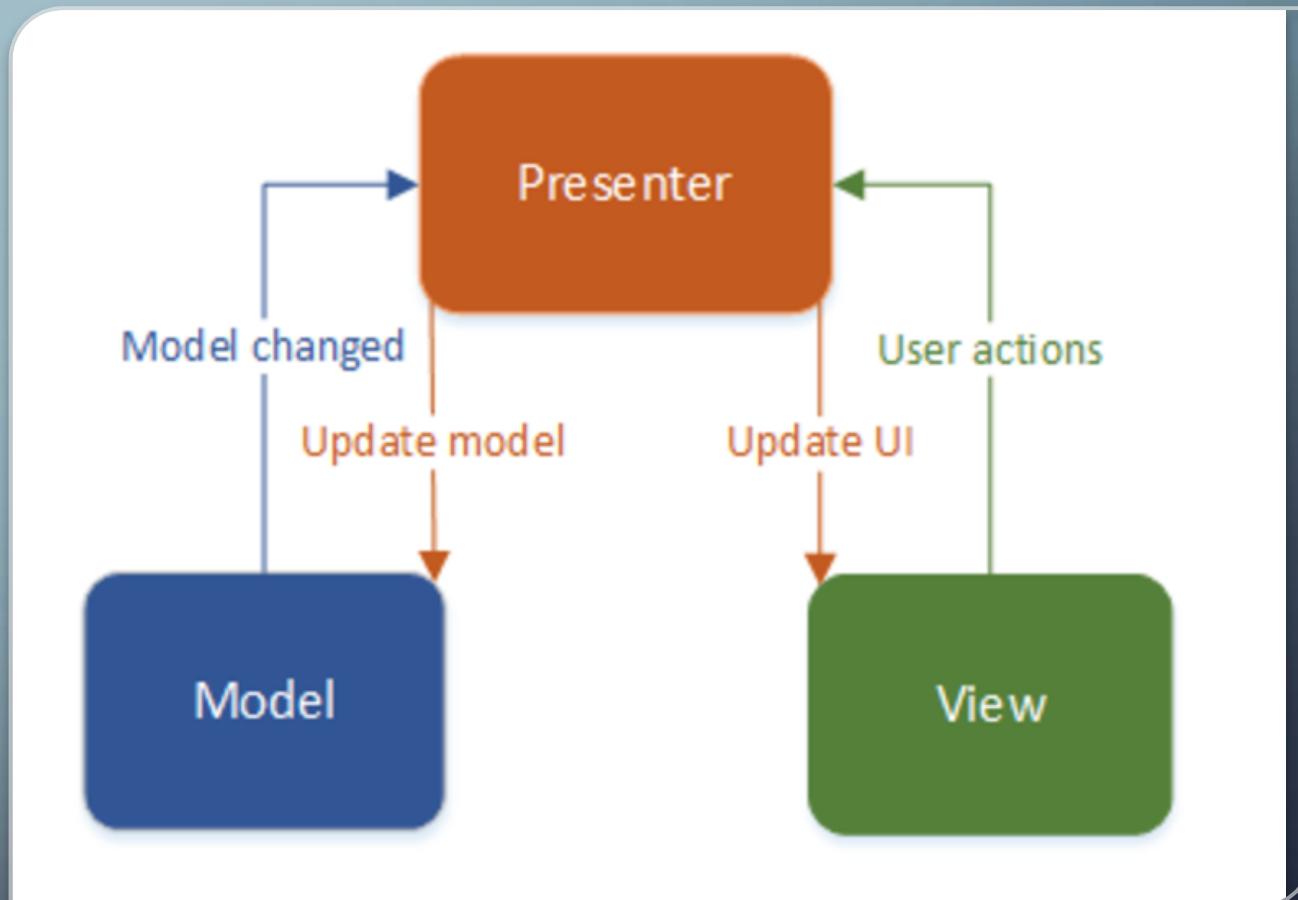


Il s'agit de pattern d'architecture conçus pour:

- réduire les codes complexes
- Séparer le code et ainsi rendre le code de l'interface utilisateur compréhensible, propre et gérable

Modèle Vue Présentation

- Crée dans les années 90 par Taligent, une jointure des entreprises Apple, IBM et Hewlett-Packard
- Utilisé depuis 2006 par Microsoft pour de nombreuses interfaces utilisateurs des frameworks .NET



DIFFÉRENCES ENTRE LE MVC ET LE MVP

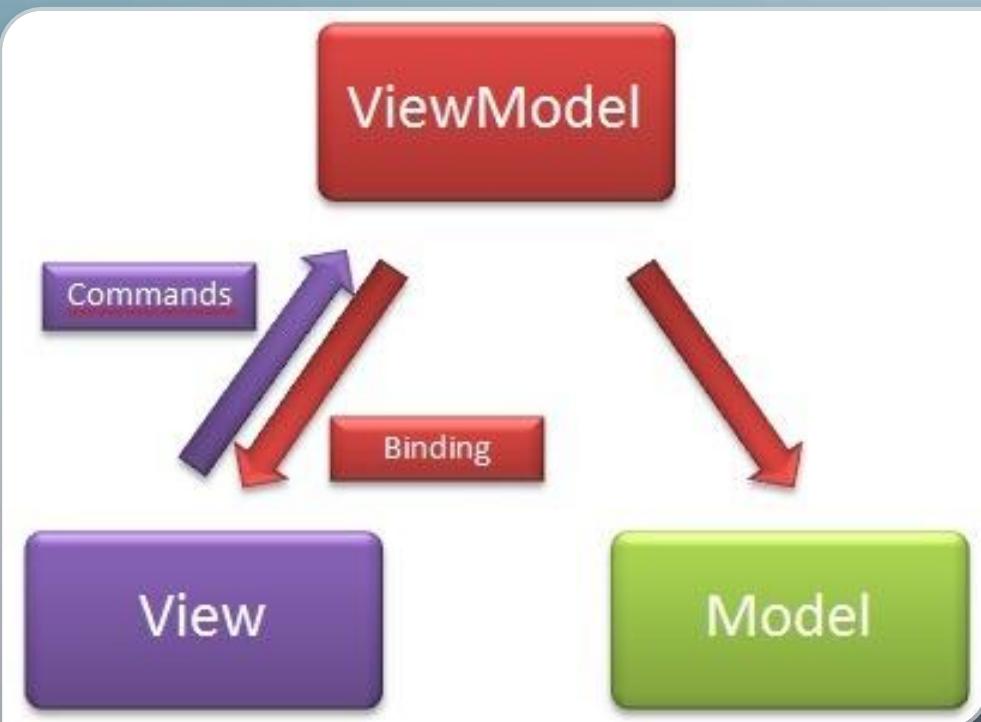
Contrôleur du MVC / Présentateur du MVP

MVC - Contrôleur	MVP - Présentateur
Ne gère que les requêtes	Contient la logique de l'application (Le modèle se comportant que comme une base de données)
La Vue interragissait avec le Modèle	La Vue n'interagit plus avec le modèle

INTÉRÊTS DU PATTERN MVP

- Découpler la Vue du Modèle
- Avoir la possibilité d'avoir plusieurs vues présentées du seul même modèle
(Table de données)
- Avoir la possibilité d'avoir une vue de plusieurs modèles différents (Vue Combinée)

MODÈLE VUE MODÈLE-VUE



- Crée à l'origine par Microsoft en 2004, pour être mieux adapté à son application Silverlight
- Même principe que celui du MVC, mais utilise le **binding** pour lier la vue et le modèle, et permettre l'évolution de celui-ci par le biais des événements

DIFFÉRENCES ENTRE LE MVC ET LE MVVM

Modèle du MVC VS. Modèle du MVVM

MVC	MVVM
Contient seulement les données	Contient l'objet entier, l'entité qui possède ses données ainsi que toutes les méthodes qui lui sont liées.

Contrôleur du MVC / Modèle-Vue du MVVM

MVC - Contrôleur	MVVM
Ne gère que les requêtes	Invoque les méthodes du modèle et transforme les données récupérées en données lisible pour la vue
Effectue la liaison entre les données et la vue	Intervient également aux appels pour les événements

COMMENT EST UTILISÉ LE MVVM ?

- Lors de la création d'interfaces graphiques lourdes, pour simplifier l'écriture (retrouvé dans la plupart des applications Microsoft)
- Lorsque les projets contiennent de très nombreuses interfaces → permet de réduire le nombre de fichiers à charger et améliore la lisibilité
- Lorsqu'on utilise des langages comme WPF, Silverlight, ou de l'Angular JS, lorsque l'on recherche un contrôle très présent dans le développement

SITOGRAPHIE

<http://ima.udg.edu/~sellares/EINF-ES1/MVC-Toni.pdf>

<https://www.codeproject.com/Articles/822791/Developing-MVC-applications-using-SOLID-principles>

<https://www.martinfowler.com/eaaDev/uiArchs.html>

<https://medium.com/@saklynejmeddinne/mcv-mvp-ou-mvvm-quel-sch>

<https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/25552-structurez-mieux-votre-code-avec-le-pattern-mvc>



QCM!

QU'EST-CE QU'UN PATTERN D'ARCHITECTURE ?

- Un plan qui permet de réaliser un gratte-ciel
- Un pattern conçu dans l'objectif d'apporter une solution à grande échelle sur la forme du logiciel
- Un pattern conçu pour résoudre des problèmes récurrents par la meilleure solution
- Un jenga de design pattern

QU'EST-CE QU'UN PATTERN D'ARCHITECTURE ?

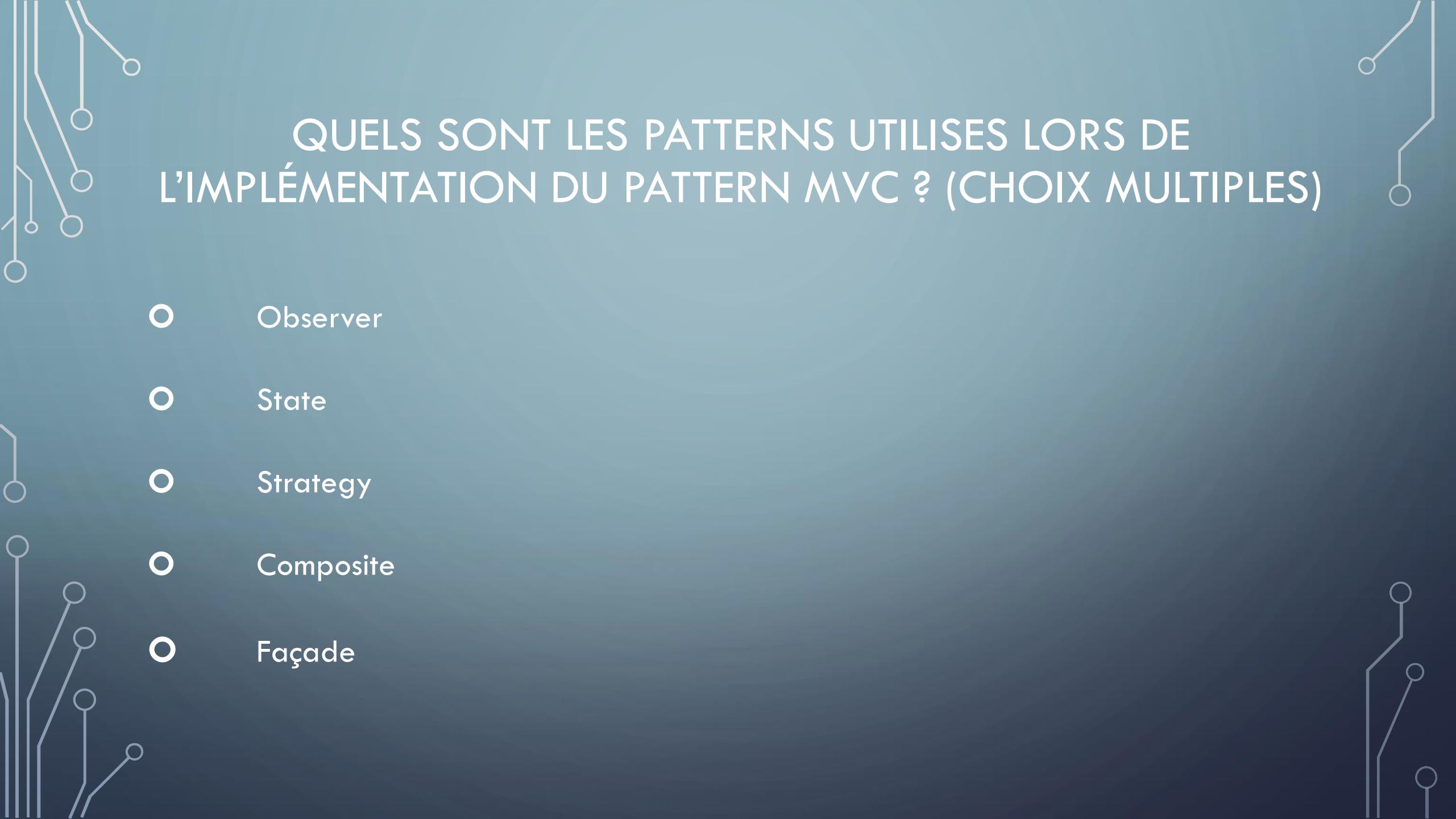
- Un plan qui permet de réaliser un gratte-ciel
-  Un pattern conçu dans l'objectif d'apporter une solution à grande échelle sur la forme du logiciel
- Un pattern conçu pour résoudre des problèmes récurrents par la meilleure solution
- Un jenga de design pattern

QUELLES SONT LES TROIS COUCHES DU PATTERN MVC ?

- Mangue, Vanille, Chocolat
- Module, Vue, Contrôleur
- Modèle, Vue, Contrôleur

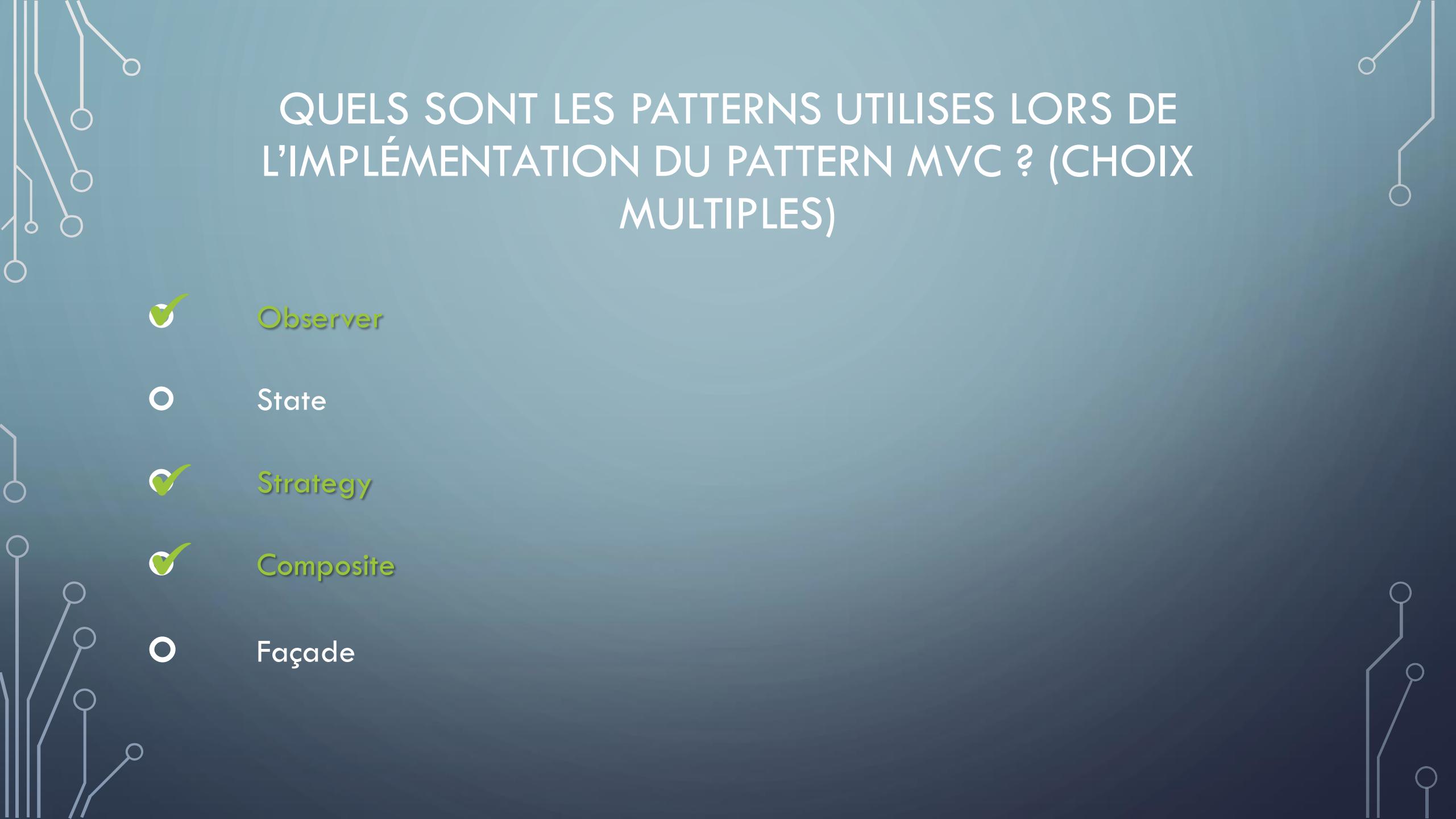
QUELLES SONT LES TROIS COUCHES DU PATTERN MVC ?

- Mangue, Vanille, Chocolat
- Module, Vue, Contrôleur
- Modèle, Vue, Contrôleur



QUELS SONT LES PATTERNS UTILISES LORS DE L'IMPLÉMENTATION DU PATTERN MVC ? (CHOIX MULTIPLES)

- Observer
- State
- Strategy
- Composite
- Façade



QUELS SONT LES PATTERNS UTILISES LORS DE L'IMPLÉMENTATION DU PATTERN MVC ? (CHOIX MULTIPLES)

- Observer
- State
- Strategy
- Composite
- Façade

QUELS SONT LE OU LES LIEN(S) ENTRE LA VUE ET LE MODÈLE ?

- Aucun, ce sont deux éléments qui n'interagissent jamais entre eux
- Le Modèle agit en fonction des actions sur la Vue
- La Vue lit les données du Modèle pour les afficher et le Modèle envoie des messages à la Vue
- C'est la VM

QUELS SONT LE OU LES LIEN(S) ENTRE LA VUE ET LE MODÈLE ?

- Aucun, ce sont deux éléments qui n'interagissent jamais entre eux
- Le Modèle agit en fonction des actions sur la Vue
- La Vue lit les données du Modèle pour les afficher et le Modèle envoie des messages à la Vue
- C'est la VM



QUEL PRINCIPE SOLID N'EST PAS TOUJOURS RESPECTÉ ?

- SRP
- OCP
- LSP
- ISP
- DIP

QUEL PRINCIPE SOLID N'EST PAS TOUJOURS RESPECTÉ ?

- SRP
- OCP
- LSP
- ISP
- DIP

QUEL PATTERN N'EXISTE PAS ?

- MVP
- MVC
- MVVM
- MisterMV

QUEL PATTERN N'EXISTE PAS ?

- MVP
- MVC
- MVVM
- MisterMV