DESIGN PATTERN

02/12/2020

PLAN DE LA FORMATION

- Introduction aux design pattern
- Factory
- Observer
- Strategy
- Singleton
- Decorator

1 INTRODUCTION AUX DESIGN PATTERN

Qu'est-ce qu'un design pattern ?

1 INTRODUCTION AUX DESIGN PATTERN

Différents problèmes de conception & Récurrents

Création des *design patterns*
(ou *masques de conceptions* en français)

Chaque design pattern répond à un problème **précis**.

1 INTRODUCTION AUX DESIGN PATTERN

Plusieurs familles de design pattern :

1. Les patterns de création
2. Les patterns de structuration
3. Les patterns comportementaux.

INTRODUCTION AUX DESIGN PATTERN

Répertoriés par le créateur Christoper Alexander
dans **A Pattern Language**
dans les années **70.**

Puis dans le livre **Design Patterns - Elements of Reusable Object-Oriented Software**
du **Gang Of Four**
composé de **Erich Gamma, Richard Helm, Ralph Johnson** et **John Vlissides**
en 1994.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Fabrique (Factory) et **Fabrique abstraite (Abstract Factory)**

Adaptateur (Adapter)

Ce patron convertit l'interface d'une classe en une autre interface exploitée par une application. Permet d'interconnecter des classes qui sans cela seraient incompatibles.

Pont (Bridge)

Ce patron permet de découpler une abstraction de son implémentation, de telle manière qu'ils peuvent évoluer indépendamment. Il consiste à diviser une implémentation en deux parties : une classe d'abstraction qui définit le problème à résoudre, et une seconde classe qui fournit une implémentation. Il peut exister plusieurs implémentations pour le même problème et la classe d'abstraction comporte une référence à l'implémentation choisie, qui peut être changée selon les besoins.

Cela ressemble à Strategy mais Bridge répond à la question « Comment construire un composant logiciel ? » alors que Strategy répond à la question « Comment voulez-vous exécuter un comportement dans un logiciel ? ».

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Monteur (Builder)

Ce patron sépare le processus de construction d'un objet du résultat obtenu. Permet d'utiliser le même processus pour obtenir différents résultats. C'est une alternative au pattern fabrique. Cet objet comporte des propriétés qui peuvent être modifiées et une méthode pour créer l'objet final en tenant compte de toutes les propriétés. Ce pattern est particulièrement utile quand il y a de nombreux paramètres de création, presque tous optionnels.

Chaîne de responsabilité (Chain of responsibility)

Le patron chaîne de responsabilité (en anglais chain of responsibility) vise à découpler l'émission d'une requête de la réception et le traitement de cette dernière en permettant à plusieurs objets de la traiter successivement. Dans ce patron, chaque objet comporte un lien vers l'objet suivant, qui est du même type. Plusieurs objets sont ainsi attachés et forment une chaîne. Lorsqu'une demande est faite au premier objet de la chaîne, celui-ci tente de la traiter, et s'il ne peut pas il fait appel à l'objet suivant, et ainsi de suite.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Commande (Command)

Ce patron emboîte une demande dans un objet, permettant de paramétrer, mettre en file d'attente, journaliser et annuler des demandes. Dans ce patron, un objet commande correspond à une opération à effectuer. L'interface de cet objet comporte une méthode execute(). Pour chaque opération, l'application va créer un objet différent qui implémente cette interface — qui comporte une méthode execute(). L'opération est lancée lorsque la méthode execute() est utilisée. Ce patron est notamment utilisé pour les barres d'outils.

Composite (Composite)

Le patron composite permet de composer une hiérarchie d'objets, et de manipuler de la même manière un élément unique, une branche, ou l'ensemble de l'arbre. Il permet en particulier de créer des objets complexes en reliant différents objets selon une structure en arbre. Ce patron impose que les différents objets aient une même interface, ce qui rend uniformes les manipulations de la structure. Par exemple dans un traitement de texte, les mots sont placés dans des paragraphes disposés dans des colonnes dans des pages ; pour manipuler l'ensemble, une classe composite implémente une interface. Cette interface est héritée par les objets qui représentent les textes, les paragraphes, les colonnes et les pages.

Décorateur (Decorator)

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Façade (Facade)

Le patron façade fournit une interface unifiée sur un ensemble d'interfaces d'un système. Il est utilisé pour réaliser des interfaces de programmation. Si un sous-système comporte plusieurs composants qui doivent être utilisés dans un ordre précis, une classe façade sera mise à disposition, et permettra de contrôler l'ordre des opérations et de cacher les détails techniques des sous-systèmes.

Poids-mouche (Flyweight)

Dans le patron flyweight (en français poids-mouche), un type d'objet est utilisé pour représenter une gamme de petits objets tous différents. Ce patron permet de créer un ensemble d'objets et de les réutiliser. Il peut être utilisé par exemple pour représenter un jeu de caractères : un objet factory va retourner un objet correspondant au caractère recherché. La même instance peut être retournée à chaque fois que le caractère est utilisé dans un texte.

Interpréteur (Interpreter)

Le patron comporte deux composants centraux : le contexte et l'expression ainsi que des objets qui sont des représentations d'éléments de grammaire d'un langage de programmation.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Itérateur (Iterator)

Ce patron permet d'accéder séquentiellement aux éléments d'un ensemble sans connaitre les détails techniques du fonctionnement de l'ensemble. C'est un des patrons les plus simples et les plus fréquents. Selon la spécification originale, il consiste en une interface qui fournit les méthodes Next et Current.

Médiateur (Mediator)

Dans ce patron il y a un objet qui définit comment plusieurs objets communiquent entre eux en évitant à chacun de faire référence à ses interlocuteurs. Ce patron est utilisé quand il y a un nombre non négligeable de composants et de relations entre les composants. Par exemple dans un réseau de 5 composants, il peut y avoir jusqu'à vingt relations (chaque composant vers quatre autres). Un composant médiateur est placé au milieu du réseau et le nombre de relations est diminué : chaque composant est relié uniquement au médiateur. Le mediator joue un rôle similaire à un sujet dans le patron observer et sert d'intermédiaire pour assurer les communications entre les objets.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

(Mémento) Memento

Ce patron vise à externaliser l'état interne d'un objet sans perte d'encapsulation. Permet de remettre l'objet dans l'état où il était auparavant. Ce patron permet de stocker l'état interne d'un objet sans que cet état ne soit rendu public par une interface. Il est composé de trois classes : l'origine — d'où l'état provient, le memento —, l'état de l'objet d'origine, et le gardien qui est l'objet qui manipulera le memento. L'origine comporte une méthode pour manipuler les memento. Le gardien est responsable de stocker les memento et de les renvoyer à leur origine. Ce patron ne définit pas d'interface précise pour les différents objets, qui sont cependant toujours au nombre de trois.

Observateur (Observer)

Prototype

Ce patron permet de définir le genre d'objet à créer en dupliquant une instance qui sert d'exemple — le prototype. L'objectif de ce patron est d'économiser le temps nécessaire pour instancier des objets. Selon ce patron, une application comporte une instance d'un objet, qui sert de prototype. Cet objet comporte une méthode clone pour créer des duplicata.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Proxy

Ce patron est un substitut d'un objet, qui permet de contrôler l'utilisation de ce dernier. Un proxy est un objet destiné à protéger un autre objet. Le proxy a la même interface que l'objet à protéger. Un proxy peut être créé par exemple pour permettre d'accéder à distance à un objet (via un middleware). Le proxy peut également être créé dans le but de retarder la création de l'objet protégé — qui sera créé immédiatement avant d'être utilisé. Dans sa forme la plus simple, un proxy ne protège rien du tout et transmet tous les appels de méthode à l'objet cible.

Singleton

Etat (State)

Ce patron permet à un objet de modifier son comportement lorsque son état interne change. Ce patron est souvent utilisé pour implémenter une machine à états. Un exemple d'appareil à états est le lecteur audio - dont les états sont lecture, enregistrement, pause et arrêt. Selon ce patron, il existe une classe machine à états, et une classe pour chaque état. Lorsqu'un événement provoque le changement d'état, la classe machine à états se relie à un autre état et modifie ainsi son comportement.

INTRODUCTION AUX DESIGN PATTERN

23 patrons de conception :

Stratégie (Strategy)

Patron de méthode (Template method)

Ce patron définit la structure générale d'un algorithme en déléguant certains passages. Permettant à des sous-classes de modifier l'algorithme en conservant sa structure générale. C'est un des patrons les plus simples et les plus couramment utilisés en programmation orientée objet. Il est utilisé lorsqu'il y a plusieurs implémentations possibles d'un calcul.

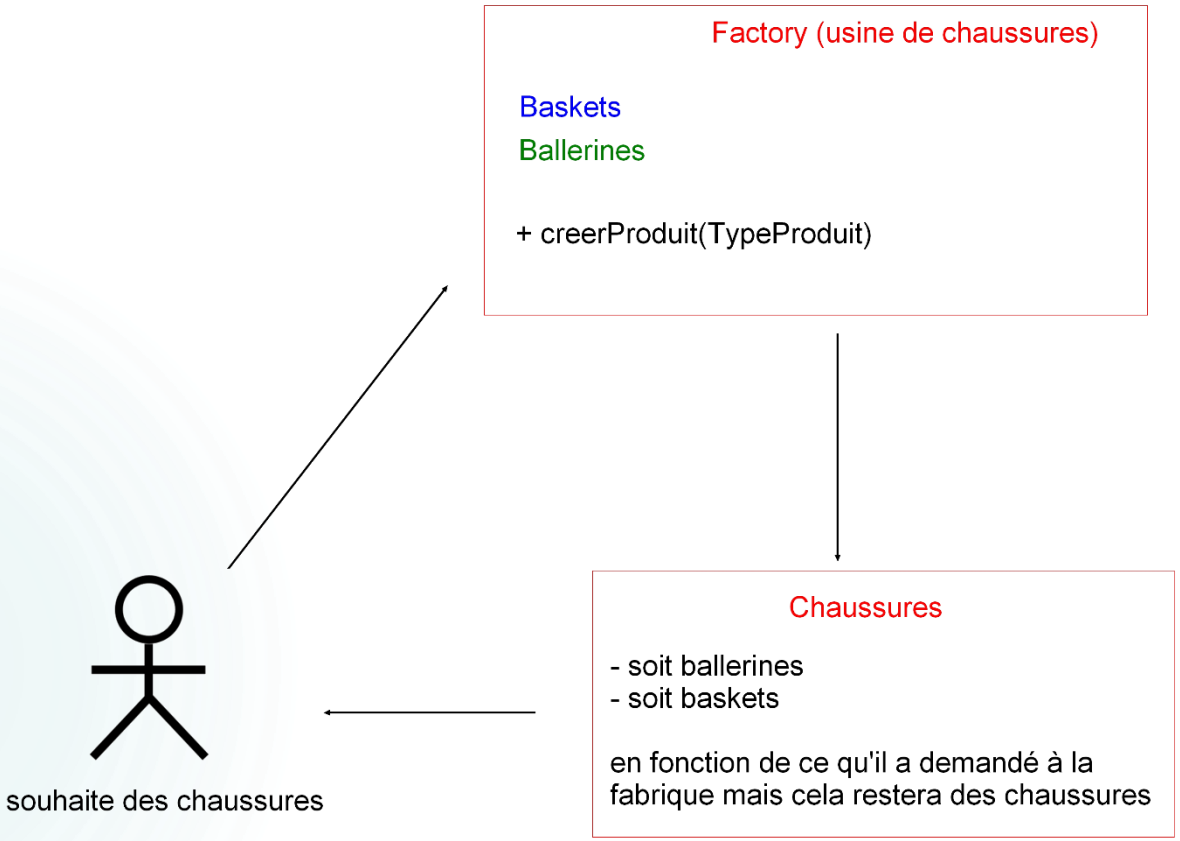
Visiteur (Visitor)

Ce patron représente une opération à effectuer sur un ensemble d'objets. Permet de modifier l'opération sans changer l'objet concerné ni la structure. Selon ce patron, les objets à modifier sont passés en paramètre à une classe tierce qui effectuera des modifications. Une classe abstraite Visitor définit l'interface de la classe tierce. Ce patron est utilisé notamment pour manipuler un jeu d'objets, où les objets peuvent avoir différentes interfaces, qui ne peuvent pas être modifiés

FACTORY (CRÉATION)

- **Comment peut-on faire pour créer plusieurs objets d'une même famille sans pour autant modifier le code ?**
- La factory permet de créer un objet dont le type dépend du contexte.
- Il faut voir cela comme une boîte qui fabrique des objets à notre demande mais nous ne savons pas comment cela s'exécute.

FACTORY



FACTORY

- Concrètement, un client souhaite des chaussures. Il demande à la factory de lui créer des chaussures (de type Baskets).
- L'usine la créé avec ses caractéristiques qui lui sont propres mais lui renvoie un objet **Chaussures**.

- S'il demande des chaussures de type Ballerines, l'usine lui renverra des **Chaussures** mais avec les caractéristiques des Ballerines.

FACTORY

Exercice

OBSERVER (COMPORTEMENT)

- **Comment certains changements d'états peuvent avoir un impact sur un ou plusieurs élément(s) sans réécrire tout le code ?**

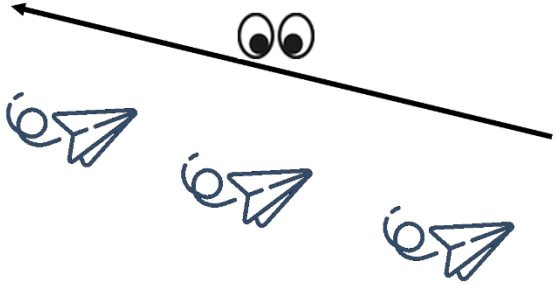
- Le design pattern Observer permet à un objet (Publisher) d'envoyer des notifications à une liste d'objets qui ont souscrit un abonnement à cet objet (Suscriber). Une fois cette notification reçue, le Subscriber peut effectuer l'action de son choix en fonction de l'information reçue.

- C'est exactement comme une newsletter. Les personnes peuvent s'abonner et reçoivent des mails lorsque des offres sont poussées par la boutique. Une fois le mail reçu et lu, vous faites ce que vous voulez : rien ou vous commandez.

OBSERVER

Etablissement 1

envoie les messages à ceux qui se sont abonnés
(=> entreprise)

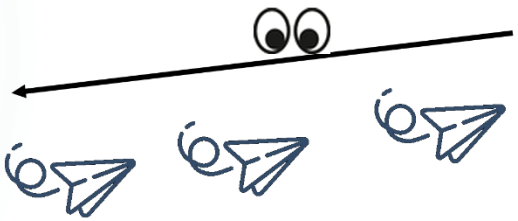


ENTREPRISE

Grâce à son abonnement sur les effectifs, entreprise sait de suite ce qu'il se passe dans les établissements
et peut agir en conséquence.

Etablissement 2

envoie les messages à ceux qui se sont abonnés
(=> entreprise)



OBSERVER

- Concrètement, une entreprise (morale) souhaite être alertée à chaque changement d'effectif car elle a un turn-over énorme.
- Pour cela, elle va « s'abonner » à la donnée Effectif de chacun de ses établissements.
- A chaque fois qu'il y a un changement dans l'Effectif d'un des établissements, cela envoie une notification à l'Entreprise.
- A aucun moment, l'établissement n'a de droit de regard sur ce que va faire l'Entreprise avec ces données. Il en fait que lui envoyer.

OBSERVER

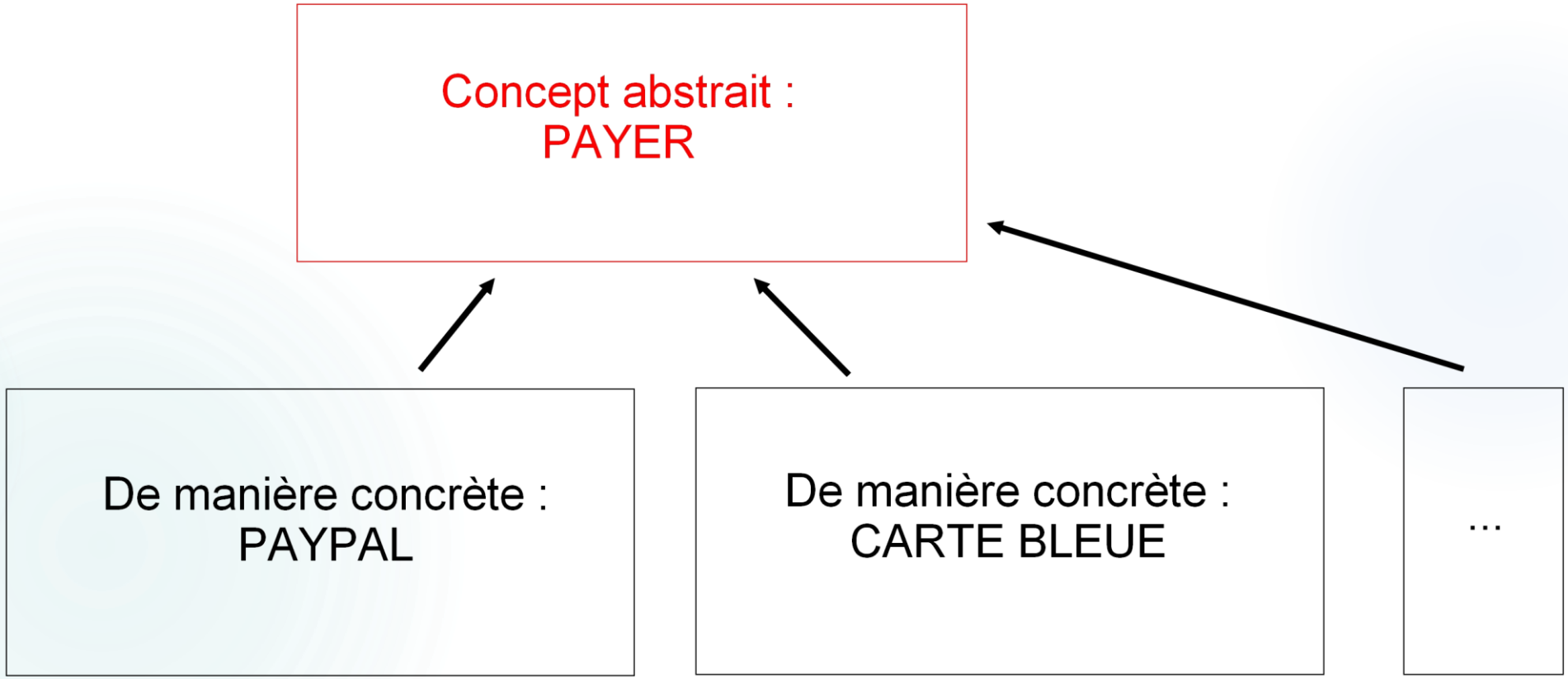
Exercice

STRATEGY (COMPORTEMENT)

- **Comment faire pour réaliser différentes opérations avec un seul et même objet ?**

- Le design pattern Strategy répond à ce problème. En effet, on crée un objet abstrait soulevant le problème et on utilise d'autres objets qui implémente cet objet afin d'exécuter l'action de manière concrète.

STRATEGY



STRATEGY

- Concrètement, vous naviguez sur un site d'e-commerce de votre choix.
- Un article vous intéresse, vous voulez l'acheter : vous l'ajoutez au panier. Vous ouvrez votre panier, et là, le site vous demande d'effectuer l'action abstraite de "payer".
- De manière concrète, il est possible de payer de différentes manières : directement par carte, par Paypal...

STRATEGY

Exercice

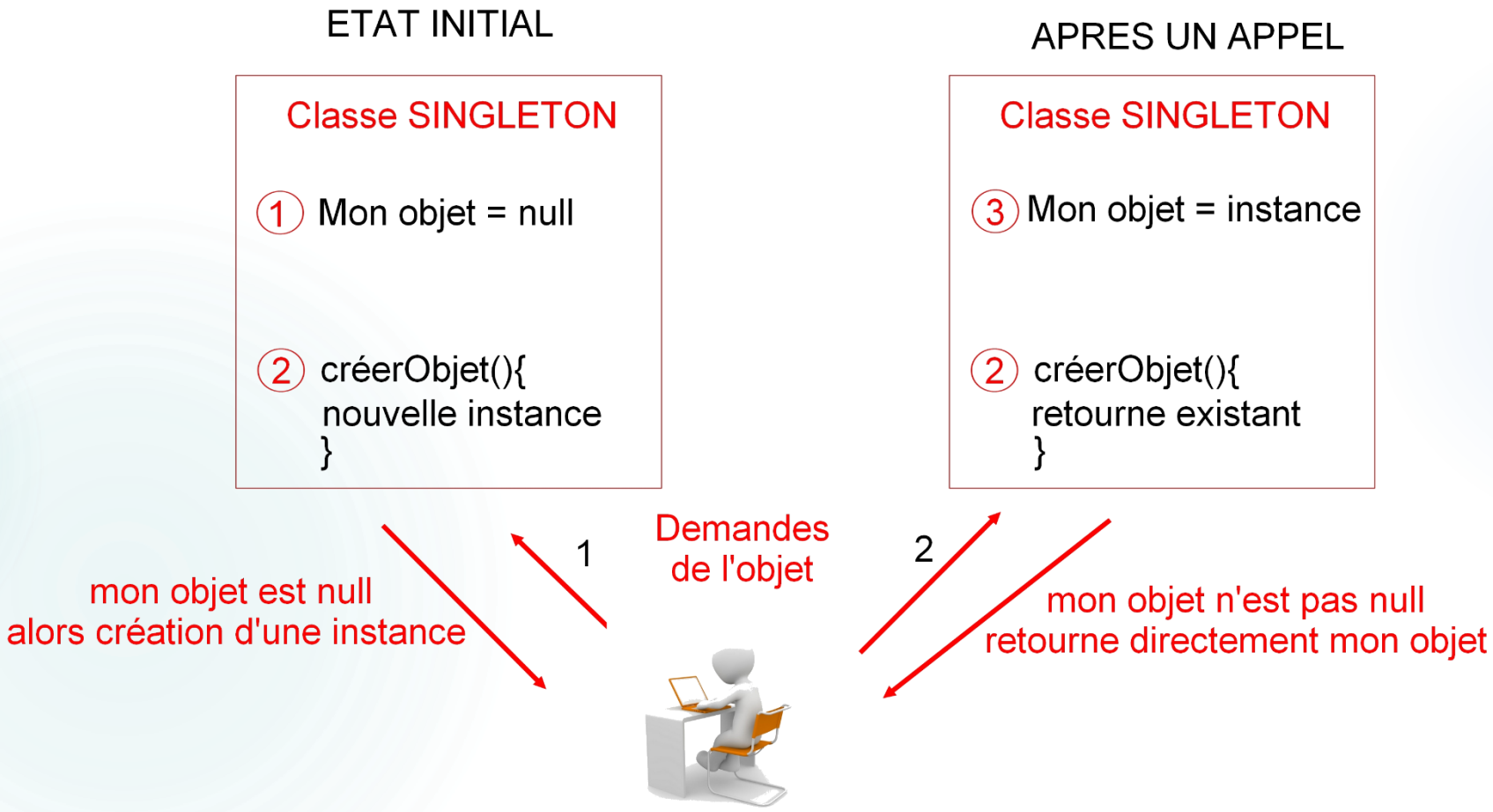
SINGLETON (CRÉATION)

- **Comment s'assurer pour qu'il n'y ait qu'une seule instance d'une classe pour tout le programme?**

- L'objet qui ne doit exister qu'en une seule instance comporte une méthode pour obtenir cette unique instance et un mécanisme pour empêcher la création d'autres instances.

- C'est un des patrons les plus simples mais les plus dangereux.
- Il ne faut pas en abuser et seulement l'utiliser lorsque **tout** le logiciel (système) en a besoin.

SINGLETON



SINGLETON

- Le classe « Singleton » possède un objet privé qui est de son propre type. Elle possède donc une instance de sa propre classe qui est null au départ.

- Lorsqu'on appelle le constructeur de la classe, ce dernier va d'abord vérifier si son propre objet n'est pas null.
- S'il ne l'est pas, il va retourner l'objet déjà créé sinon, il va créer une nouvelle instance.

SINGLETON

Exercice

DECORATOR

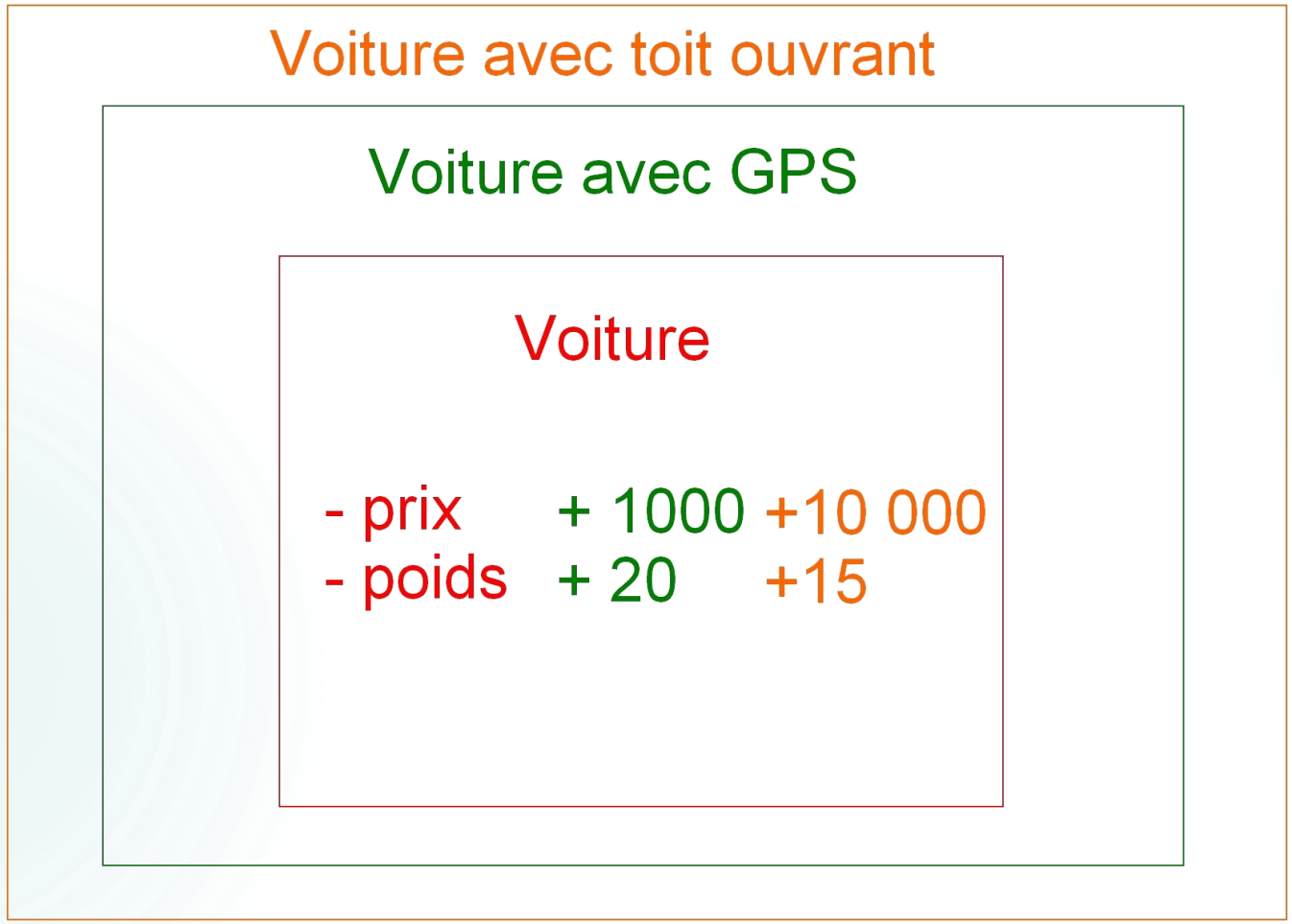
- **Comment faire pour ajouter des fonctionnalités de façon dynamique à un de nos objets sans utiliser autant de classes ?**
- Nous souhaitons rajouter plusieurs options à nos voitures. Notre concessionnaire vend des DS, des C3, C5,… & des Renegade.
- S'il souhaite proposer l'option GPS, il faudrait créer la voiture basique DS, C3, C5, Renegade et la voiture avec GPS :
  - DS avec GPS
  - C3 avec GPS
  - C5 avec GPS
  - Renegade avec GPS

DECORATOR

- Juste pour 4 voitures, cela ferait 8 classes pour répondre à un seul problème : ajouter un GPS à des voitures « basiques ».

- Imaginez avec d'autres options & d'autres voitures. C'est là que le pattern Decorator entre en jeu.

DECORATOR



DECORATOR

- Ici, nous avons une classe Voiture basique ayant un prix et un poids.
- Nous allons utiliser les classes Decorator VoitureAvecGPS & VoitureAvecToitOuvrant afin de rajouter des options sur la voiture de base.
- Ces options ont un impact sur les attributs de base (ici prix et poids).

DECORATOR

Exercice

WEBOGRAPHIE

https://www.jmdoudoux.fr/java/dej/chap-design-patterns.htm

https://blog.sodifrance.fr/design-pattern-observer-pratiquer-en-mieux-le-design-observer-en-java-8/

https://www.codingame.com/

https://fr.wikipedia.org/wiki/Patron_de_conception

https://openclassrooms.com/fr/courses/6810956-ecrivez-du-code-java-maintenable/6926911-creez-des-objets-avec-les-design-patterns-de-creation

https://design-patterns.fr/decorateur-en-java