**UNIVERSITE CHEIKH ANTA** DIOP **DE DAKAR**

**Ecole Supérieure Polytechnique de Dakar**

**Département Génie Informatique**

****

**Le Pattern Observer**

Présenté Par :

Khardiatou Basse

Marième Aidara

Pape Amadou Sakho

**SOMMAIRE**

[**Les Patterns de Comportement** 3](#_Toc86438883)

[Introduction 3](#_Toc86438884)

[Le Pattern Observer 3](#_Toc86438885)

[1. Définition et principe 4](#_Toc86438886)

[2. Les avantages et les inconvénients de l’Observer design pattern 4](#_Toc86438887)

[3. Pattern Observer : Cas d’utilisation 5](#_Toc86438888)

[4. Mise en œuvre de ce pattern 5](#_Toc86438889)

[4.1. Problème posé 5](#_Toc86438890)

[4.2. Solution 6](#_Toc86438891)

# **Les Patterns de Comportement**

## Introduction

Un patron de comportement permet de résoudre les problèmes liés aux comportements, à l'interaction entre les classes.

Les différents patrons de comportement sont les suivants :

* **Chaine de responsabilité** qui permet de construire une chaîne de traitement d'une même requête.
* **Commande** qui permet d’encapsule l'invocation d'une commande.
* **Interpréteur** quipermetd’interpréter un langage spécialisé.
* **Itérateur** qui permet de parcourir un ensemble d'objets à l'aide d'un objet de contexte (curseur).
* **Médiateur** qui permet de réduire les dépendances entre un groupe de classes en utilisant une classe Médiateur comme intermédiaire de communication.
* **Memento** qui pour rôle de Mémoriser l'état d'un objet pour pouvoir le restaurer ensuite.
* **Observateur** qui permet d’intercepter un évènement pour le traiter.
* **Etat** permet de gérer différents états à l'aide de différentes classes.
* **Stratégie** qui permet de changer dynamiquement de stratégie (algorithme) selon le contexte.
* **Patron de méthode** qui permet de définir un modèle de méthode en utilisant des méthodes abstraites.
* **Visiteur**  qui fait le découpage des classes et traitements, afin de pouvoir ajouter de nouveaux traitements sans ajouter de nouvelles méthodes aux classes existantes.

Notre rapport se portera sur le pattern Observateur (Observer)

## Le Pattern Observer

## Définition et principe

Observer est un patron de conception qui permet de mettre en place un mécanisme de souscription pour envoyer des notifications à plusieurs objets, au sujet d’événements concernant les objets qu’ils observent. L’objet que l’on veut suivre est en général appelé sujet ou observable, mais comme il va envoyer des notifications pour prévenir les autres objets dès qu’il est modifié, nous l’appellerons diffuseur. Tous les objets qui veulent suivre les modifications apportées au diffuseur sont appelés des souscripteurs ou observateurs. Observer nous propose d’ajouter un mécanisme de souscription à la classe diffuseur pour permettre aux objets individuels de s’inscrire ou se désinscrire de ce diffuseur. Les notions d'observateur et d'observable permettent de limiter le couplage entre les modules aux seuls phénomènes à observer. Le patron permet aussi une gestion simplifiée d'observateurs multiples sur un même objet observable. Dans la pratique, quand un événement important arrive au diffuseur, il fait le tour de ses souscripteurs et appelle la méthode de notification sur leurs objets. Les applications peuvent comporter des dizaines de classes souscripteur différentes qui veulent être tenues au courant des événements qui affectent une même classe diffuseur.

## Les avantages et les inconvénients de l’Observer design pattern

L’utilisation de l’Observer pattern dans le développement logiciel peut s’avérer avantageuse dans de nombreuses situations. Le principal avantage de ce concept est le **haut degré d’indépendance** entre l’objet observé (sujet) et les observateurs qui s’appuient sur le statut actuel de cet objet. Par exemple, l’objet observé n’a nullement besoin de disposer d’informations sur ses observateurs puisque l’interaction peut être réalisée indépendamment via l’interface de l’observateur. Les **observateurs reçoivent les mises à jour automatiquement** ce qui supprime totalement les requêtes sans résultat puisque le sujet reste le même.

Cependant, le fait que le sujet informe automatiquement tous les observateurs enregistrés des modifications ne présente pas que des avantages : en effet, les**informations sur la modification** sont transmises même si elles ne sont**pas pertinentes** pour l’un des observateurs. Cela peut se révéler problématique si le nombre d’observateurs enregistrés est très important puisque le modèle Observateur pourra gaspiller un temps de calcul considérable dans la transmission. Un autre problème du patron de conception Observateur est que, bien souvent, le code source du sujet ne permet pas de déterminer quel observateur doit recevoir les informations.

## Pattern Observer : Cas d’utilisation

L’Observer design pattern est tout particulièrement utile dans les applications basées sur des composants dont le statut est

* d’une part, fortement observé par les autres composants et
* d’autre part, soumis à des modifications régulières.

Parmi les cas d’utilisation typiques, on trouve les IGU (Graphical User Interfaces), des interfacesservant à la communication avec un logiciel et permettant une utilisation facile par les utilisateurs. Dès que des données sont modifiées, elles doivent être actualisées dans tous les composants de l’IGU, une tâche qui peut être assurée de façon optimale grâce à la structure sujet/observateur de l’Observer pattern. Les programmes utilisant des ensembles de données à visualiser (tableaux classiques ou diagrammes graphiques) profitent également de l’organisation effectuée par ce modèle de conception.

En principe, l’Observer design pattern ne comporte aucune restriction en ce qui concerne le langage de programmation utilisé. Pour que l’implémentation de ce modèle soit pertinente, il suffit que le paradigmeorientéobjet soit supporté. Parmi les langages dans lesquels l’utilisation de ce patron est très appréciée, on trouve notamment C#, C++, Java, JavaScript, Python et PHP.

## Mise en œuvre de ce pattern

## Problème posé

Nous mettons en place une application de conversion en plusieurs bases. Le principe de notre application c’est de convertir automatiquement un nombre entré en base décimale par l’utilisateur en base binaire, base octale et base hexadécimale.

Et nous devons disposer d’un convertisseur pour chacune de ses bases. Notre problème est comment signaler automatiquement à ces trois convertisseurs qu’il y a un nouveau nombre à convertir pour qu’il effectue la conversion.

## Solution

La réalisation de cette application nécessite de créer une classe abstraite Observer contenant une méthode modifier (), dont vont héritées les trois concrètes BinaryObs, OctaleObs et HexObs qui vont définir la méthode modifier ().

On disposera aussi d’une classe **Theme** qui va avoir tous ces observateurs dans un tableau, qui sera notre observable et qui notifiera tous ces convertisseurs c’est-à-dire les observateurs de l’arrivée d’un nouveau nombre entré à convertir.