**Pattern OBSERVER**

Design pattern de comportement : c’est quoi ?

* description de structures d’objets ou de classes avec leurs interactions
* Deux types de motifs
* Motif de comportement de classes :
* utilisation de l’héritage pour répartir les comportements entre des classes (ex : Interpreter)
* Motifs de comportement d’objets avec l’utilisation de l’association entre objets :
* pour décrire comment des groupes d’objets coopèrent (ex : Mediator)
* **pour définir et maintenir des dépendances entre objets (ex : Observer)**
* pour encapsuler un comportement dans un objet et déléguer les requêtes à d’autres objets (ex : Strategy, State, Command)
* pour parcourir des structures en appliquant des comportements (ex : Visitor, Iterator)

Le pattern **OBSERVER** c’est quoi ?

1. **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.

**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.