|  |  |
| --- | --- |
| **Nom du collège:** | **Collège de Rosemont** |
| **Nom des étudiants:** | **Raphael Albern** |
| **Numéro de groupe:** | **38425** |
| **Nom du cours:** | **Qualité des applications** |
| **Numéro du cours:** | **420-B34-RO** |
| **Objectif:** | **Design Pattern - Visiteur** |

**PATRON - VISITEUR**

**Ce quoi un patron de conception?**

Un patron de conception est une solution répétable pour des problèmes généraux.

**Description du patron de conception - visiteur**

* Le patron Visiteur est dans la catégorie de patrons comportementaux c’est à dire qu’il s’intéresse à la communication et interaction entre les classes et leurs objets.
* L’objectif du patron Visiteur est de permettre de définir une nouvelle opération pour une hiérarchie sans changer ses classes.
* Le patron visiteur est utilisé lorsqu’on doit effectuer la même action sur plusieurs objets de types différents.
* Le patron Visiteur permet de séparer de manière claire un algorithme d’une structure de donnée. L’algorithme n’est donc pas implémenté dans la classe mais dans des classes externes.
* On limite ainsi tout couplage entre les données et leurs traitements ce qui permet d’ajouter des traitements à nos structures de données sans avoir à les modifier.  De plus la classe externe est informé du type exact de l’objet qu’il visite à l’aide du principe du double dispatch.

**Implémentation du patron visiteur**

La classe visitable (c’est à dire la structure de donnée) possède une méthode **accept** qui nous permet d’injecter notre visiteur.

En même temps les visiteurs concret implémentent une interface qui contient une méthode **visit** par type d’objet visitable (ainsi on se sert du typage dynamique pour lors de l’utilisation appellera la bonne méthode – double dispatch).

**Avantage**

* Le pattern visitor permet de définir une nouvelle opération pour une hiérarchie  
  sans avoir à modifier ses classes.
* Le pattern visitor permet de **centraliser le code fonctionnel**. L’implémentation correspondant à une opération est centralisée dans la classe visiteur et n’a pas à être complétée dans les autres classes individuelles.
* On n’a pas besoin d’aller dans chaque classe pour faire la modification de méthodes.

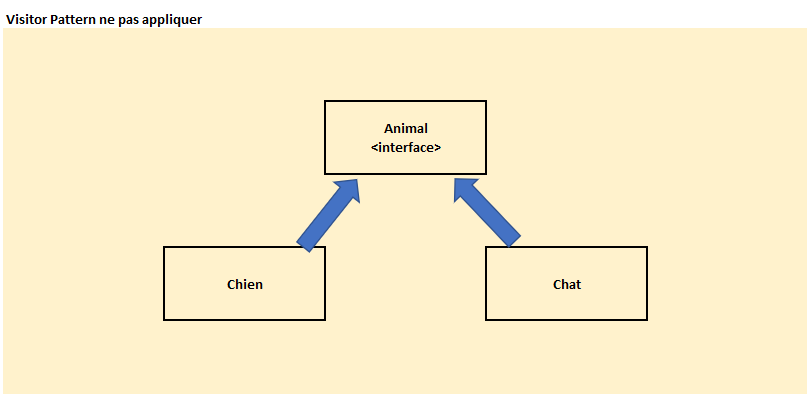
**Inconvénient :**

* Toutes les modifications apportées à la classe d’un élément, même les plus petites, entraînent généralement un ajustement des classes visiteurs affectées
* Le pattern visitor rend la structure plus complexe.

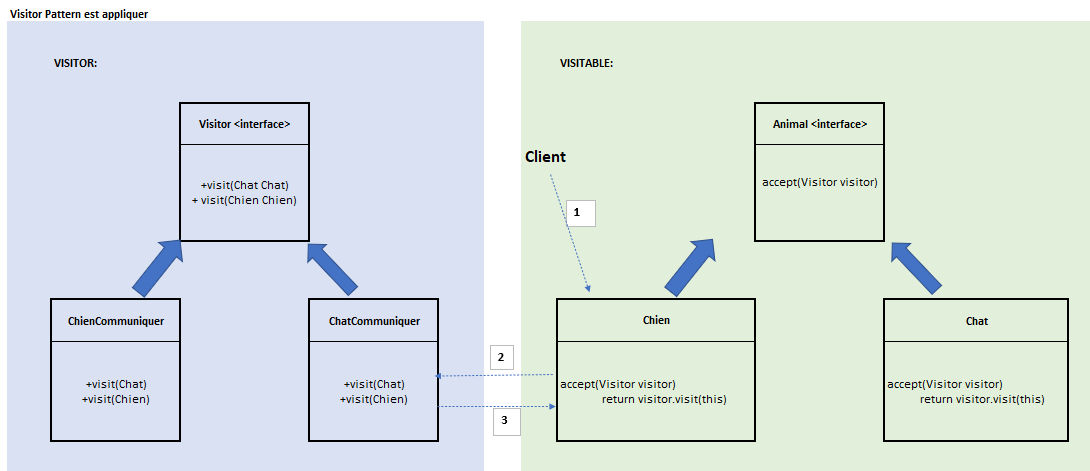
**Exemple de code – Exemple pour voir le problème que le Visitor Pattern résoudre**

Voir les codes – Exemple\_Animal\_sansVisiteur

Cet exemple démontre le problème qu’on peut faire face parce que Java ne supporte pas Double Dispatch (c’est-à-dire : ils ne peuvent pas déduire dynamiquement le type des arguments des méthodes)

****

Voir les codes – Exemple\_Animal\_avecVisiteur

****

**Exemple de code Visitor:**

Voir les codes – Exemple\_Visiteur\_Tax

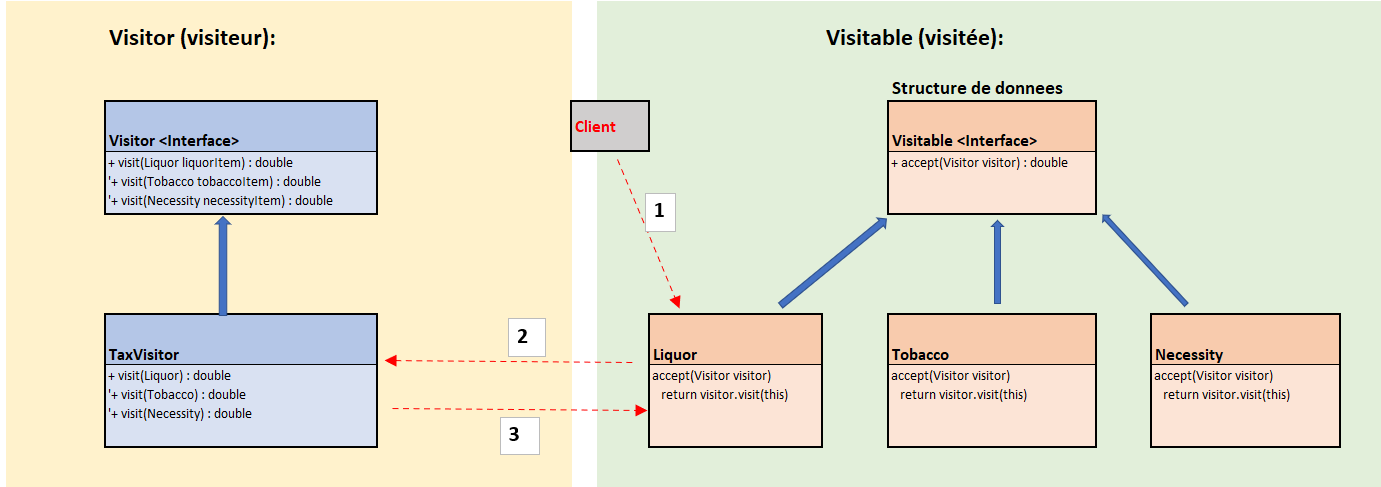
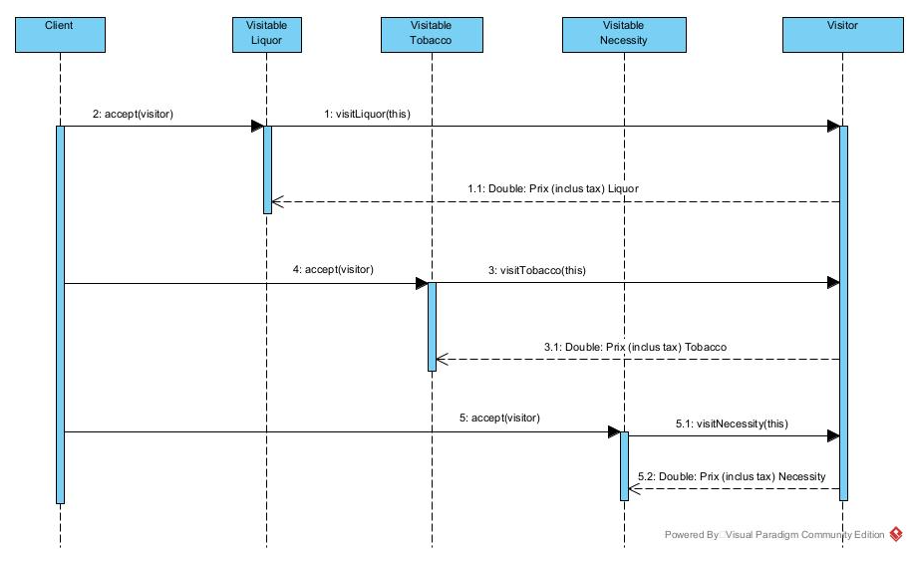
****

Diagramme de séquence :

****