LaFac.com

BOSOM - CHAUSSY - DELLOUVE

20 décembre 2013

Introduction

Afin de travailler la conception de systèmes réels, nous avons créé un modélisé un pseudo-site de vente en ligne.

L'implémentation s'est faite en Java et nous devions répondre à plusieurs contraintes, à savoir un système de gestion de personnes, de produits et de différentes offres applicables.

Diagramme de classe

La classe principale de notre projet est la classe Contexte. Il s'agit d'un **Singleton**, c'est-à-dire qu'il n'en existe qu'une seule instance dans le programme. Ce contexte représente, en quelques sorte, la base de donnée que possèderait un site réel. Si nous devions nous passer de celui-ci, nous basculerions les offres employés dans le statut *Employe* qui deviendrait alors un **Singleton**. Nous ajouterions aussi un autre **Singleton** qui contiendrait les *OffreFidelites* pour les *Adherents*.

Parmi les autre **Design Pattern** utilisés nous avons le design **State** pour les statuts des *Personnes*. En effet, nous devions laisser la possibilité aux *Personnes* de changer de *Statut* en cours d'exécution, rendant ce design indispensable.

Un autre **Design Pattern** que nous utilisons est celui de l'**Observer**. Il consiste en une classe qui en "observe" une autre. Ainsi, lors d'un changement de l'instance de cette classe, l'**Observer** associé effectuera une suite d'actions prédéfinies. Dans notre projet, la classe *Alerte* est un **Observer** sur la classe *Panier*. Lors d'un changement de celui-ci, l'*Alerte* affichera un message.

Enfin nous avons fait le choix d'une liaison faible entre les *Produits* et les *Offres* afin que la modification d'une offre n'impacte pas le produit en lui même. En effet, une offre doit connaitre le produit sur lequel elle s'applique, mais le contraire n'est pas vrai. De plus, cela évite qu'un changement dans les *Offres* ne produise une erreurs ailleurs, améliorant la robustesse de notre programme.

La généricité du langage Java est utilisée lors des affichages. En effet, chaque type de *Produit* à un affichage personnalisé, tout comme les *Offres*, qui est géré lors de l'exécution du programme. Celui-ci va chercher la bonne fonction à appeler et la définir dans la classe abstraite force la redéfinition de celle-ci dans les sous-classes.

Hook (le crochet) n'est pas un DP mais un idiome. Il consiste à définir une méthode template dans la classe du haut et à laisser les sous-classes redéfinir les méthodes virtuelles qui les concernent. C'est quand même une bonne habitude de programmation qui peut être citée ici, par exemple pour implémenter l'affichage des offres en laissant chaque classe afficher les spécificités (nombre de produits, ...) qui les concernent.

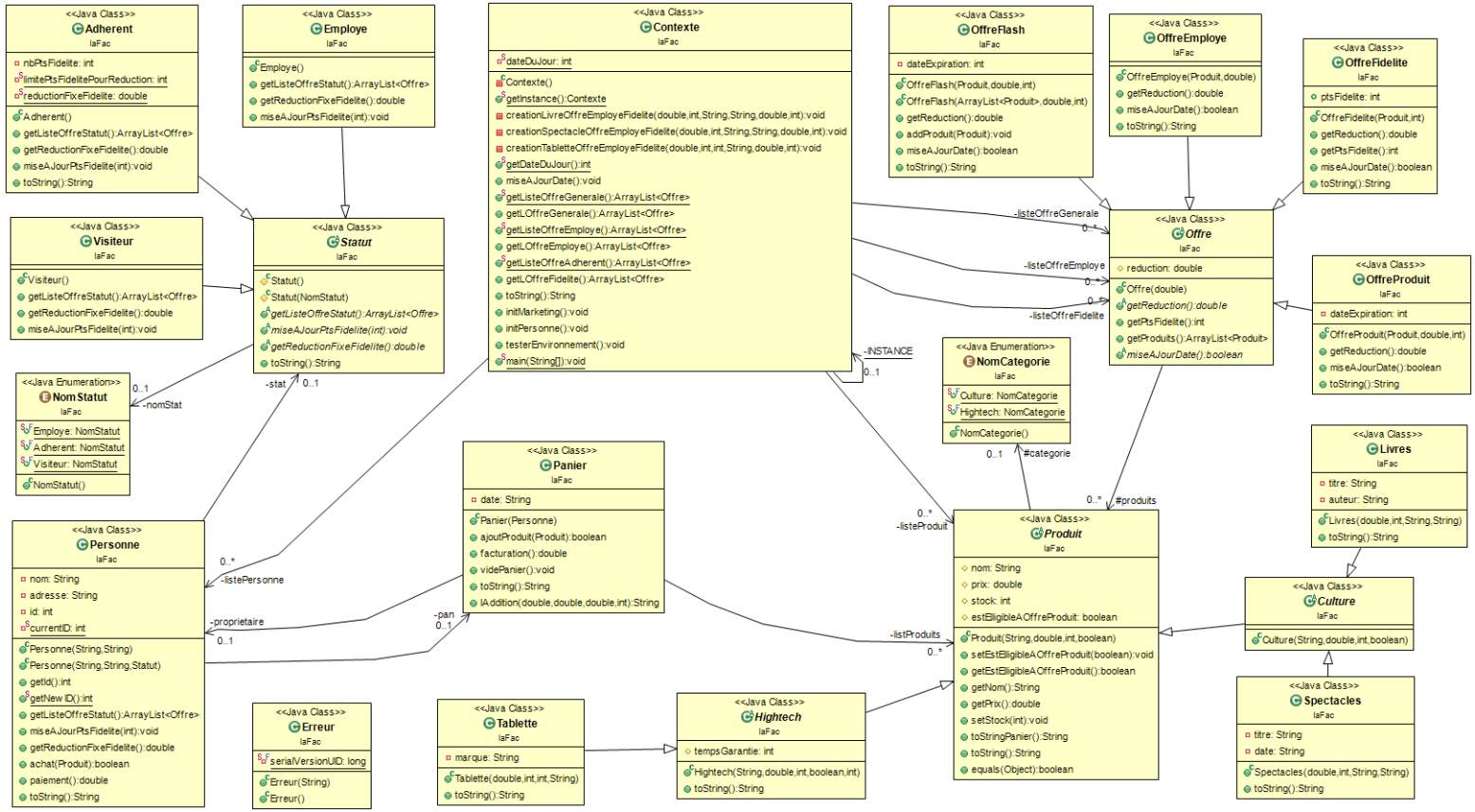

FIGURE 1 – Diagramme de classe

Diagrammes de séquences

Conclusion

Ce projet nous a permis de réfléchir à un problème concret, mettant en œuvre les connaissances acquises durant le cours.

Nous avons utilisé plusieurs **Design Pattern**, afin de répondre aux différentes contraintes du sujet, et la généricité du langage afin que les interactions entre les objets soient le plus simple et le plus clair possible, tout en offrant une possibilité d'évolution du code.

En résumé, il s'agissait d'un exercice enrichissant qui nous aura permis d'acquérir de nouvelles connaissances.

Annexe - Code source