Design patern : Visitor

Isli Zaoui Mathieu Laisné

December 2022

1 Cas d'utilisation abstrait

Nous disposons de classes et nous cherchons à effectuer un traitement similaire à chacune d'elle. Le problème premier qui se pose est que ces classes ne nécessitent pas l'exact même traitement et qu'il nous est donc impossible de créer une méthode de traitement unique, il n'est pas non plus recommandé par nos collègues de modifier lourdement le code de ces classe en y ajoutant une méthode propre à chacune, et enfin il n'est pas envisageable d'implémenter une vérification de la classe car elles sont unies sous une même interface et cette solution impliquerait du code lourd avec des *cast* partout.

2 Exemple

Un restaurant cherche à implémenter une méthode de calcul de prix des commandes passées. Il faut prendre en compte le fait que la commande soit sur place ou en livraison, si la commande est un menu elle dispose d'une réduction et si la boisson commandée est alcoolisée il y a un supplément à payer. Dans un souci de code propre et facile à maintenir on se propose d'implémenter le patron **Visitor**. Celui-ci permet d'externaliser un traitement similaire mais tout de même spécifique à chacune des classes traitées via un *visiteur* qui concentre le code d'intérêt (calculer le prix). Ce visiteur est ensuite utilisé via une méthode simple (communément appelée *accept*, ici *calculatePrice*) qui fait le lien entre les sous classes et le visiteur.