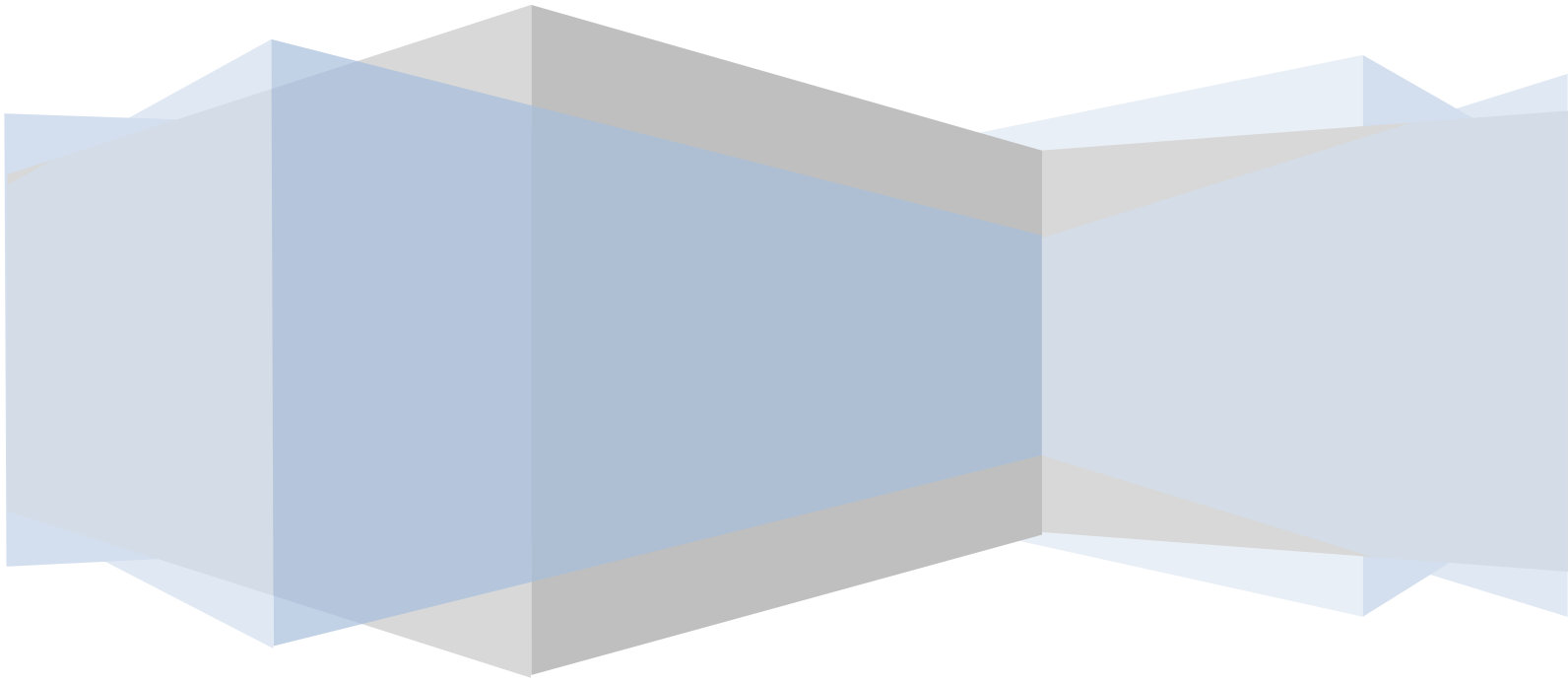
RAPPORT DE TP 3

GUEDDAL Sarra

BOUDAOUD nesrine

RSI A−B



INTRODUCTION:

scenario :

Il y a un café bien connu, qui a un menu traditionnel, mais au cours de la dernière période, le responsable du marketing a constaté que le nombre de personnes qui visitent le café a diminué.

Il a donc décidé de résoudre ce problème et d'ajouter au menu un grand nombre de variétés comme le lait, le moka, les chocolats, le soja, etc.

Donc, pour cet exemple, nous avons le café : est une classe abstraite.

et le menu comporte 4 types de café : Americano, Espresso, Cappuccino ...puis il a ajouté toutes les nouvelles options comme une sous-classe sous Coffee SuperClass..comme Americanowithmocha ou Americanowithmilk ... et la même chose pour les autres sous-classes.

les problèmes de cette implémentation:

-c'est une implémentation surchargée et condensée.

-Si vous voulez ajouter une nouvelle catégorie ou un nouveau type de café, vous devez créer votre propre classe et l'implémenter.

-Que se passe-t-il si le prix du lait est modifié ?! vous changerez la méthode de calcul des coûts dans toutes les classes de café. (ouvert à la modification) .

Voyons comment le design pattern va résoudre ce problème:

Tout d'abord, nous devons nous familiariser avec un principe de conception très important dans le logiciel. Il s'agit du deuxième principe de SOLID pour l'uncle Bob.

Principe d'ouverture/fermeture(open closed /principal) : les classes doivent être ouvertes à l'extension, mais fermées à la modification.

Ensuite, nous allons définir le modèle Decorator, qui provient du modèle de conception de la structure.

Modèle de conception des décorateurs : Attacher des responsabilités supplémentaires à un objet de manière dynamique. Les décorateurs fournissent une alternative flexible à la sous-classification pour étendre la fonctionnalité.

ce qui signifie que vous pouvez attacher une fonction spécifique ou un comportement supplémentaire à votre objet - appliquons cela à notre coffeeshop

notre UseCase : nous voulons calculer le coût d'une tasse de café americano avec double Mocha, lait et chocolat.

Comment allons-nous faire ?

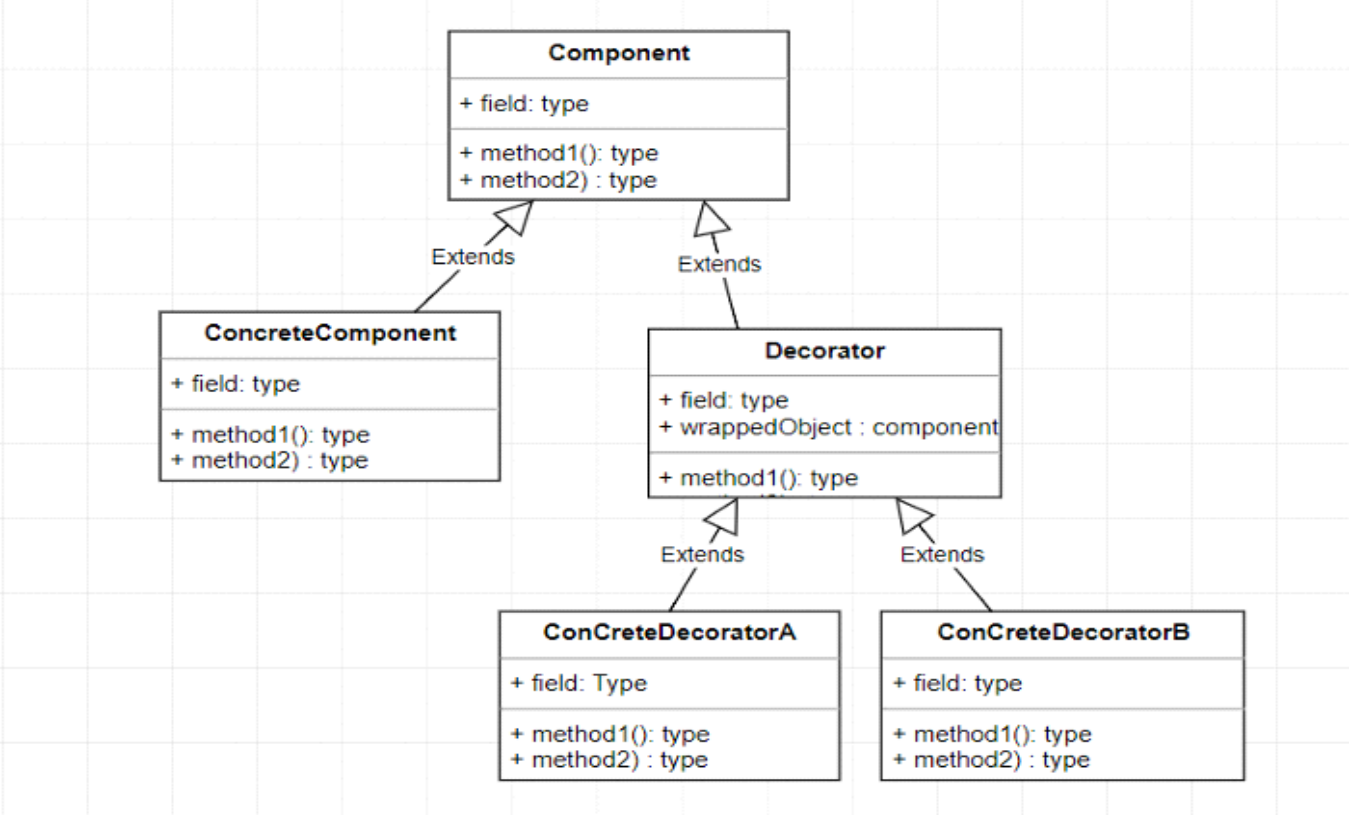
- Prenez la sous-classe Americano coffee.

- Décorez-le avec du moka, puis décorez-le à nouveau avec du moka.

- Décorez-la avec du lait, puis décorez-la avec du chocolat.

- appelez cost() de ce composant pour le calculer en ajoutant les coûts des autres éléments de décoration.

L'implémentation de base de ce design pattern :



- component can be used by each own or wrapped by a decoration.

- ConcreteComponent: it is the subclass that can be decorated by any decoration item.

- Decorator implements same interface or abstract class as the component it is going to decorate.

- each decorator HAS-A (wraps) a component, which means that the decorator hold a reference variable of the component that is going to decorate it.

- peut être utilisé seul ou enveloppé par une décoration.

- ConcreteComponent : c'est la sous-classe qui peut être décorée par n'importe quel élément de décoration.

- Le décorateur implémente la même interface ou classe abstraite que le composant qu'il va décorer.

- Chaque décorateur est enveloppé par un composant, ce qui signifie que le décorateur contient une variable de référence du composant qu'il va décorer.

Note : CoffeeDecorator est une classe abstraite qui étend la classe abstraite Coffee.

Note : si une classe abstraite étend une autre classe abstraite, il est possible d'écraser les méthodes abstraites de la classe parente mais dans tous les cas, il n'y aura pas d'erreur de compilation.

Note : nous avons respecté les principes SOLID par cette implémentation en :

1- nous n'avons pas modifié la classe parente du café.

2- il est simple d'ajouter un nouveau décorateur comme (soja) ou d'ajouter un nouveau type de café en étendant les classes parentes et sans les modifier donc c'est ouvert pour l'extension, mais fermé pour la modification.

Decorateur :

Décorateur est un patron de conception structurel qui permet d’affecter dynamiquement de nouveaux comportements à des objets en les plaçant dans des emballeurs qui implémentent ces comportements.

Il permet de décorer le comportement d'une classe. et il Permet l'habillage dynamique d'objets afin de modifier leurs responsabilités et comportements existants Le modèle Decorator atteint un seul objectif : ajouter dynamiquement des responsabilités à tout objet.

Principaux cas d'utilisation :

-Ajouter des fonctionnalités/responsabilités supplémentaires de manière dynamique.

-Supprimer des fonctionnalités/responsabilités de façon dynamique.

-Eviter trop de sous-classes pour ajouter des responsabilités supplémentaires.

Avantages et Inconvénients:

La prise en compte du patron Décorateur lors de la conception d’un logiciel est payante pour plusieurs raisons. Tout d’abord, il y a le haut degré de flexibilité qui vient avec une telle structure de décorateur : tant au moment de la compilation qu’à l’exécution, les classes peuvent être étendues avec de nouveaux comportements sans héritage. Cette approche de programmation n’entraîne pas de hiérarchies d’héritage floues, ce qui améliore également la lisibilité du code du programme.

Le fait que la fonctionnalité soit répartie entre plusieurs classes de décorateurs augmente également la performance du logiciel. Ainsi, vous pouvez appeler et lancer les fonctions dont vous avez besoin pour le moment. Avec une classe de base complexe, qui fournit toutes les

fonctions en permanence, cette option optimisée en termes de ressources n’est pas disponible.

Cependant, le développement selon le patron Decorator n’a pas que des avantages : avec l’introduction du modèle, la complexité du logicielaugmente automatiquement. L’interface de Decorator en particulier est généralement très verbeuse et associée à de nombreux nouveaux termes, et donc tout sauf facile d’accès pour les débutants. Un autre inconvénient est le grand nombre d’objets Decorator, pour lesquels une systématisation séparée est recommandée afin d’éviter d’être confronté à des problèmes de vue d’ensemble, similaires à ceux rencontrés lors du travail avec des sous-classes. Les chaînes d’appel souvent très longues des objets décorés (c’est-à-dire les composants logiciels étendus) rendent également plus difficile la recherche d’erreurs et donc le processus de de débogage en général.

Probleme :

Dans notre cas, lors de l’envoi d’un message dans une communication entre deux personnes sur internet,Il faut assurer les conditions de securité :

-Confidentialité

-Intégrité

-Authentification

On essaie dans ce TP de les assurer a l’aide de patron décorateur.

Avant de faire le chiffrement,On a rajouté les classes suivantes :

-Majuscule,

-Eliminer des espaces,

-Eliminer la ponctuation

Pour faciliter le chiffrement ,et minimiser l’intervalle de code ASCII.

Puis ,Les classes suivantes :

-MD5,SHA Pour assurer l’intégrité.

-chiffrement avec decalage,pour assurer la confidentialité.

Toutes les classes doivent héritées de la classe abstraite
« Decorateur »,qui elle-même doit implemente l’interface « Imessage ».

Donc, On peut décorer le message à envoyé avec une hiérarchie de ces décores,et a la réception de ce message ces classes assurent l’inverse de ces fonctionnalité :le déchiffrement…pour pouvoir les lire en clair.