LOG2410
TP4 - Réponses aux questions

Par :
- Lotfi Meklati (1953909)
- Simon Kaplo (1947701)

Session : Hiver 2020
Groupe : 01

3.1) Identifiez ’intention du patron Decorator.

L’intention du patron décorateur est d’ajouter dynamiquement de nouvelles fonctionnalités à un objet individuel sans modifier l’implémentation de sa classe.
Dans notre cas la classe FixedRebate permet d’appliquer un rabais au coût d’un objet, pour ce faire, elle réutilise les méthodes computeCost() et computeMarginalCost() des objets qu’elle décore pour retourner une nouvelle valeur avec le rabais appliqué.
Ce patron décorateur évite ainsi de créer une méthode “calculer rabais” pour tous les composants qui héritent de AbsSubscriptionPlan.

4.1) Identifiez l’intention et les avantages du patron Visiteur.

L'intention du patron visiteur est de définir une opération qui va être appliqué sur les différent élément d’une structure d’objet sans avoir à modifier la classe de ces éléments.
Dans notre cas la classe VariableRebate permet d'appliquer un calcul de rabais différent en fonction du type de forfait. Cela est fait grâce à la fonction processVariableRebate() qui appelle la méthode accept() qui va exécuter un code d’initiation de rabais différent dépendamment des classes OccasionalPlan, RegularPan et PremiumPlan.
La méthode accept() appelle à son tour une des méthodes process spécifiques à chaque classe, qui va alors permettre de connaître le rabais à appliquer.
L’usage du patron Visiteur nous donne un avantage en terme de flexibilité car VariableRebate est indépendante des classes d’objet c’est à dire qu’une opération dans Visiteur comme par exemple computeCost() fonctionne indépendamment de la classe OccasionalPlan, RegularPlan ou PremiumPlan.
De plus l’usage du patron Visteur nous donne l’avantage d’avoir une fonctionnalité localisée car tout le code associé à la fonctionnalité du calcul du rabais est dans un seul endroit bien identifié.

4.3) Si en cours de conception, vous constatez que vous devez ajouter une nouvelle sous-classe dérivée de AbsSubscriptionPlan, établissez la liste de toutes les classes qui devraient être modifiées.

Les classes AbsSubscriptionPlanVisitor et VariableRebate, pour ajouter une méthode process qui puisse assurer une double invocation de la nouvelle classe.

4.4) Comment se comportent les objets impliqués dans le calcul du prix de l'énergie si les deux types de décorateurs sont appliqués simultanément sur un même forfait ? Que se passe-t-il si on applique 2 rabais variables sur le même forfait ?

Si on applique VariableRebate sur un FixedRebate, le VariableRebate va visiter le plan décoré par FixedRebate pour savoir quel rabais appliquer. Cela se fait grâce à la fonction processFixedRebate() qui oblige le plan décoré à accepter VariableRebate.

```
void VariableRebate::processFixedRebate(FixedRebate& rebate) {
    // To be completed
    rebate.getPlan().accept(*this);
}
```

Si on applique FixedRebate sur un VariableRebate, le FixedRebate va faire appel à getSubscriptionCost() de la classe AbsRebateDecorator.

```
Amount AbsRebateDecorator::getSubscriptionCost() const
{
    return m_plan.getSubscriptionCost();
}
```

Or, comme m_plan est un VariableRebate et que celui hérite aussi du patron décorateur, la fonction getSubscriptionCost() va être appelée une deuxième fois récursivement de façon à avoir accès aux informations de la classe visitée par VariableRebate. C'est ce qui permet à la classe FixedRebate de connaître le coût de l'abonnement.

Lorsqu'on applique un VariableRebate sur un autre VariableRebate, le deuxième VariableRebate va visiter le plan visité par le premier VariableRebate pour savoir quel rabais appliquer. Cela se fait grâce à la fonction processVariableRebate().

```
void VariableRebate::processVariableRebate(VariableRebate& rebate) {
    // To be completed
    rebate.getPlan().accept(*this);
}
```