R6.A.06

TP2 : Refactoring to patterns

L'objectif de ce TP est d'appliquer des refactogins à une application dans l'objectif de l'améliorer en appliquant un patron de conception.

Exercice 1 : Retrouver un patron dans un code

L'application proposée un petit extrait du framework de test JUnit, c'est pourquoi le code n'est pas complet. Elle est utilisée pour implémenter des suites de tests pour des programmes.

Sur moodle récupérer le code dans test.zip

1) Faire la rétro-conception pour obtenir le diagramme de classes de l'application.
2) Quel patron de conception semble être utilisé dans cette application ? Identifier le rôle des classes de l'application par rapport à la définition de ce pattern.
3) Proposer une amélioration classique pour ce patron que vous ajouterait dans le code et dans le diagramme.

Rendre :

- Le code source final
- le diagramme de classes
- La réponse aux question 1 et 2.

Exercice 2 : Application du pattern Strategy

Quand une application utilise beaucoup de logiques conditionnelles dans un calcul, l'objectif du refactoring est de déléguer le calcul à un objet Strategy.

Le code de l'exemple concerne le calcul du capital pour des prêts bancaires. Il ne montre qu'une bonne partie de la logique conditionnelle utilisée pour effectuer ce calcul, bien qu'il s'agisse d'une logique encore moins conditionnelle que celle contenue dans le code d'origine, qui devait gérer les calculs de capital pour 7 profils de prêt différents.

A faire ;

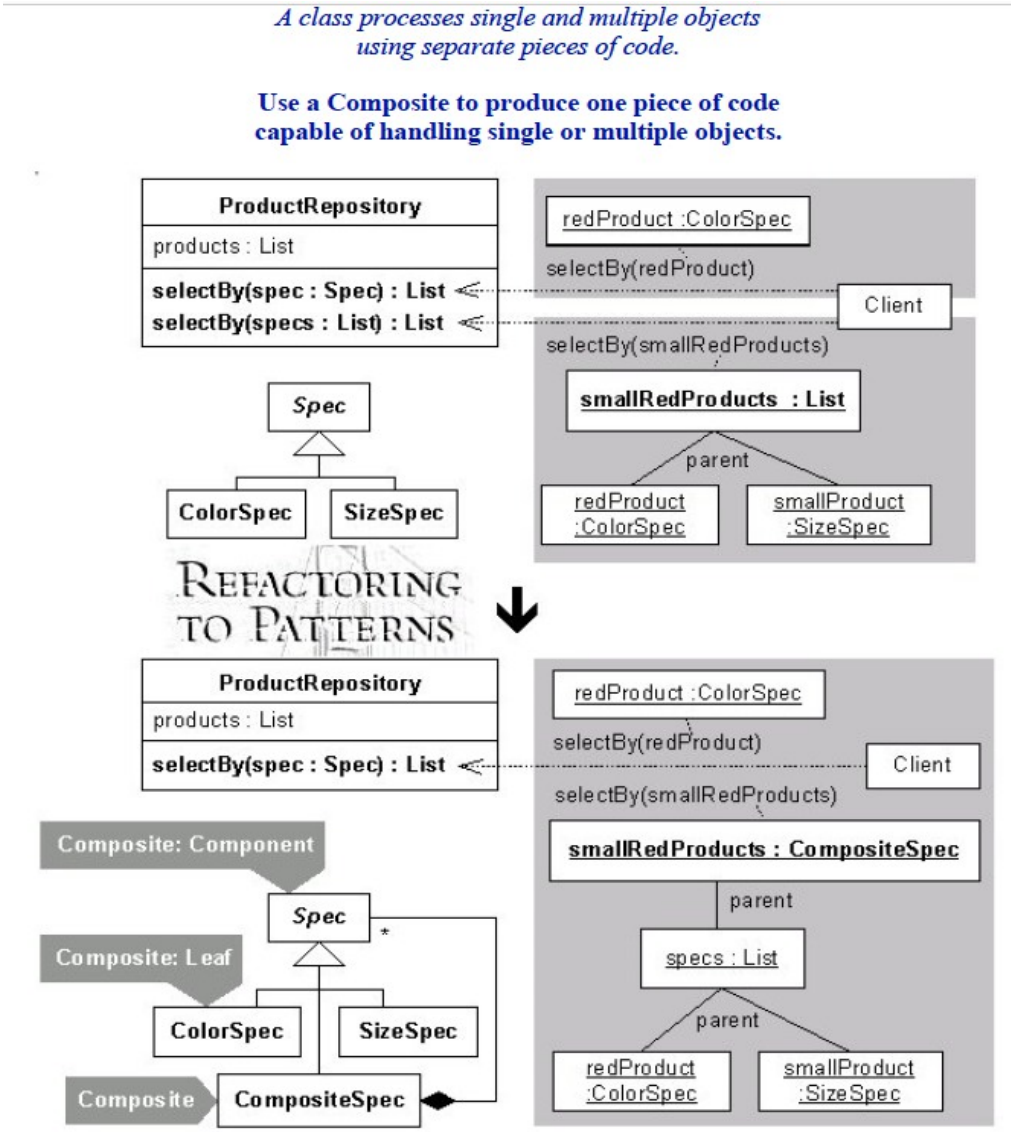
1 . Récupérez le code de la classe `Loan`, et le fichier strategy-mechanics
La mécanique pour « **Remplacer des calculs conditionnels avec Strategy »** est donné dans le fichier « strategy-mechanics ».
2. Rédiger les 9 étapes de la mécanique (en les adaptant à l'exemple proposé) et donner les codes intermédiaires que vous aurez produits (étape 6 et étape 9).
A la fin (étape 9) produisez également le diagramme de classes de la version finale.

Rendre :

- Le code source final
- le diagramme de classes final
- La rédaction des 9 étapes

Exercice 3 : Replace One/Many distinctions with Composite

Ici on vous donne une solution pour le refactoring et on vous demande d'expliquer la démarche sur l'exemple pour trouver la solution.



Le même exemple un peu plus précis est donné sur la page suivante.

Rendre :

Les explications de la démarche


ProductRepository
-products : List
+selectBy(spec : ProductSpecification) : List
+selectBy(productSpecs : List) : List
Client
Client
ProductSpecification redProducts =
new ProductColorSpecification(Color.red));
List foundProducts =
productRepository.selectBy(redProducts);
ProductSpecification
+isSatisfiedBy(Product) : boolean
ProductColorSpecification
ProductSizeSpecification
List smallRedProducts = new ArrayList();
smallRedProducts.add(
new ProductSizeSpecification(ProductSize.SMALL));
smallRedProducts.add(
new ProductColorSpecification(Color.red));
List foundProducts =
productRepository.selectBy(smallRedProducts);
ProductRepository
-products : List
+selectBy(spec : ProductSpecification) : List
Client
Client
ProductSpecification redProducts =
new ProductColorSpecification(Color.red));
List foundProducts =
productRepository.selectBy(redProducts);
2
ProductSpecification
+isSatisfiedBy(Product) : boolean
ProductColorSpecification
ProductSizeSpecification
AndProductSpecification
+AndProductSpecification(
ProductSpecification augend,
ProductSpecification addend)
+isSatisfiedBy(Product) : boolean
ProductSpecification smallRedProducts =
new AndProductSpecification(
new ProductSizeSpecification(ProductSize.SMALL),
new ProductColorSpecification(Color.red)
);
List foundProducts =
productRepository.selectBy(smallRedProducts);
public boolean isSatisfiedBy(Product product) {
return augend.isSatisfiedBy(product)
&& addend.isSatisfiedBy(product);
}