CNAM

M-NFA032 Algorithmique et programmation : bibliothèques et patterns

Troisième partie

Travail à effectuer

Rappels

Quelques éléments importants avant de commencer le fil rouge :

- à terme, votre programme comportera une interface graphique ; cette interface ne vous permettra pas de lire au clavier ni d'écrire à l'écran ; donc, ne pas mettre d'ordre de lecture ou d'écriture dans vos classes ; si vous souhaitez valider vos classes, prévoyez alors une petite bibliothèque temporaire pour les entrées-sorties.
- Dans vos classes, mettez systématiquement vos champs private ou, à la limite, protected ; vous pourrez ensuite donner des accès à vos variables d'instances via des getters et des setters.

Exercice 1 : Verrouillage d'un Lot

Un lot est la somme d'une certaine quantité de certains produits. Pour les modéliser, nous allons utiliser dans un lot, une HashMap dont la clé sera un Produit et la valeur la quantité de ce produit contenu dans le lot.

Les produits seront ajoutés un à un dans le lot mais, une fois au catalogue, le contenu du lot ne pourra plus être modifié. Pour résoudre ce problème nous allons utiliser un booléen verrouille dans lot, initialisé à false. Lot offrira un service verrouiller () qui fera basculer ce booléen à false et qui ne pourra plus être modifié par la suite. Ce service sera appelé plus tard quand on ajoutera le produit au catalogue.

La démarche à suivre est la suivante :

1. Dans la classe Produit, définissez une méthode verrouiller sans paramètre, ne retournant pas de résultat et ne faisant rien ; c'est le comportement normal pour la majorité des produits ; seul, les lots ont un comportement différent que nous allons définir par la suite ;
2. Dans la classe Lot, ajoutez une variable d'instance private nommée « verrouille » de type booléen ;
3. Dans le constructeur de Lot, initialisez la variable verrouille définie ci-dessus à false ;
4. Redéfinissez la méthode verrouiller dans Lot :
    1. menu Source > Override/Implement Methods
    2. sélectionnez verrouiller
    3. Ok
5. Remplacez les deux lignes insérées par Eclipse par une simple affectation forçant la valeur de verrouiller à true

Exercice 2 : Contenu d'un Lot

Comme annoncé précédemment, pour définir un lot, on créera une instance de Lot vide, puis on ajoutera successivement chacun des composants du lot. Une fois la liste des composants finie, le lot sera ajouté au catalogue qui verrouillera le lot avant de l'enregistrer.

Nous allons utiliser une HashMap pour mémoriser la composition d'un lot. La clé sera le produit et la valeur, la quantité de ce produit. Pour faire au plus simple, nous nous contenterons de simplement fournir un service d'ajout d'un produit. Cette classe pourra être complétée au besoin.

Notez enfin que nous sommes en train d'appliquer un patron de conception « Adaptateur ».

A réaliser

1. Ajouter une variable d'instance « contenu » de type HashMap dans la classe Lot. Le type de la clé sera Produit et celui de la valeur, Integer.
2. Prévoir l'initialisation de la variable d'instance soit dans la ligne de déclaration, soit dans le constructeur. Initialisation à une nouvelle instance de la classe HashMap...
3. Prévoir dans Lot une méthode « ajouter », prenant en paramètre un produit et une quantité et ajoutant ces informations à contenu.
    1. Si le lot est verrouillé, il faudra lever une exception spécifique. La définition de l'exception sera incluse dans la classe Lot.
        1. Ajoutez en début de la méthode ajouter le code suivant :

            ```
            if (verrouille)
                 throw new LotVerrouilleException();
            ```

            Une erreur doit apparaître sur LotVerrouilleException.
        2. Amenez le curseur sur LotVerrouilleException et attendez quelques secondes que la fenêtre « Quick fixes » apparaisse
        3. Choisissez Create class …
        4. Dans la nouvelle fenêtre, cochez la boîte « Enclosing type » pour inclure la définition de la classe dans Lot.
        5. Cliquez sur le bouton Finish.
        6. Une ligne import … a été ajoutée en début de fichier. Elle présente un avertissement. Cette ligne n'est pas nécessaire. Vous pouvez la supprimer en faisant Ctrl-Shift-O.
        7. La définition de la classe a été aussi ajoutée au fichier. Il doit y avoir un avertissement sur LotVerrouilleException. Pour le faire disparaître, amenez le curseur de la souris sur LotVerrouilleException. Patientiez quelques secondes puis choisissez « Add default serial version ID ». Nous verrons ce que cela signifie lors du prochain cours.
        8. Il y a une nouvelle erreur sur la ligne throw new LotVerrouilleException();. L'exception LotVerrouilleException est une extension de Exception. Les exceptions de ce type doivent être déclarées dans l'entête de la méthode. Ce concept sera abordé en NFA032 bientôt. En attendant, amenez le curseur de la souris sur l'erreur, patientez, puis cliquez sur « Add throws declaration » et, enfin, validez la proposition faite.
    2. Il sera bon aussi de vérifier que le produit n'est pas déjà présent dans le lot. Si un tel

cas se produit, lever une exception spécifique. Pour ce faire, reproduisez ce que vous venez de faire ci-dessus.

Exercice 3 : Catalogue

Nous allons commencer à créer la classe Catalogue. Elle sera complétée plus tard au fil des besoins. Pour l'instant, nous allons nous contenter de définir quelques fonctions essentielles et d'y appliquer le patron de conception « singleton » ne permettant d'avoir qu'une seule instance de Catalogue.

Le catalogue ne devant pas contenir deux produits identiques (même référence), nous pouvons utiliser un HashSet<Produit> pour stocker les produits.

A réaliser

1. Définir un package magasin destiné à recevoir les structures de données élaborées.
2. Y définir une classe Catalogue simplement
3. Appliquer le Patron de conception Singleton en ajoutant les lignes suivantes :
    1. Le patron de conception vient d'être appliqué et tout le code nécessaire a été ajouté :

        ```
        private static Catalogue instance;
        ```

        une variable statique et privée, nommée instance de type Catalogue ; cette variable mémorisera l'instance créée dès que ce sera fait, sinon elle vaudra null par défaut signifiant ainsi qu'aucun catalogue n'a encore été créé ;

        ```
        private Catalogue () {}
        ```

        un constructeur private sans paramètre et qui ne fait rien ; ceci a pour but d'interdire à l'utilisateur de la classe d'en créer lui-même une instance l'obligeant ainsi à utiliser les services que nous fournissons ;

        ```
        public static Catalogue getInstance () {
              if (instance == null)
                    instance = new Catalogue();
              return instance;
        }
        ```

        un service getInstance sans paramètre qui, si aucune instance n'a encore été créée, en créera une, et, dans tous les cas, retournera l'unique instance de Catalogue ; le seul moyen que l'utilisateur de cette classe aura pour récupérer le catalogue est cette méthode qui assure qu'une seule instance de Catalogue n'a été créée ;

4. Créez une variable d'instance private nommée « contenu » de type HashSet<Produit>
    1. Pensez à initialiser cette variable soit dans sa déclaration soit dans le constructeur défini ci-dessus
5. Ajouter une méthode d'instance pour ajouter un produit au catalogue.
    1. pensez à refuser la valeur null en levant une exception ; vous pouvez utiliser l'exception NullPointerException prédéfinie en Java ;
    2. déléguez simplement l'opération à HashSet.add et contrôlez le résultat. Si la méthode retourne false, levez une exception spécifique.
    3. pensez à verrouiller le produit si tout s'est bien passé !

6. Ajouter une méthode d'instance pour supprimer un produit du catalogue. Comme la précédente, une exception sera levée si le produit est null. Une autre exception sera levée si le produit n'est pas présent.
7. Créer une méthode qui cherche un produit dans le catalogue à partir de sa référence. Si aucun produit n'a cette référence, null sera retourné.

   Vous pourrez ajouter à la classe abstraite Produit, une méthode aPourReference (String reference) retournant true ou false et en faire usage.

Exercice 4 : Bonus

Eventuellement allez sur ce site : http://whoopdicity.blogspot.fr/2009/09/creating-singleton-using-eclipse-code.html pour trouver comment utiliser Eclipse pour ne plus avoir à refaire toutes les étapes pour appliquer un PDC Singleton à une classe.