CNAM

M-NFA035 Algorithmique et programmation : bibliothèques et patterns

Première partie

Travail à effectuer

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 : TypeProduits

Construire une énumération[1] avec, comme constantes, les différents types de produits. Elle nous servira de fabrique plus tard. Nous l'appellerons « TypeProduit » par la suite.

Méthode sous Eclipse

1. Créez un projet Java « Fil Rouge »
2. Créez une nouvelle Enum (File>New>Enum)
   1. Dans Package mettez produits
   2. Name => TypeProduit
   3. Finish
3. Entre les accolades, ajoutez les différents types de produits définis dans le cahier des charges séparés par des virgules. Exemple : STYLO, CRAYON, ...

Exercice 2 : Produit

L'objet de cet exercice est de créer une classe abstraite « Produit » qui « factorise » tous les éléments communs aux produits. Elle sera complétée plus tard

1 Une énumération est un nouveau type dont on donne la liste des valeurs possibles. Les valeurs doivent être des textes qui seront, par convention, en majuscules.

Méthode sous Eclipse

1. Dans votre Projet créez une nouvelle classe (File>New>Class)
    1. Package => produits
    2. Name => Produit
    3. Cochez abstract
    4. Finish
2. Ajoutez tous les champs communs à tous les produits (mettez les tous private et tous sauf prix final[2]). Exemple private final String reference ;
    1. Référence
    2. Marque
    3. Prix
    4. Intitulé
    5. Description

Tous les champs sauf prix doivent être soulignés en rouge car ils ne sont pas initialisés par le constructeur par défaut. Prix doit être souligné en jaune car il n'est pas utilisé. Nous corrigerons cela plus loin.

3. Ajoutez un champ private et final « type » de type TypeProduit.

Encore un champ en erreur car non initialisé. Nous allons corrigé les erreurs dans l'étape suivante en créant un constructeur.

4. Créer un constructeur avec tous les champs sauf reference qui sera généré automatiquement plus tard :
    1. Menu Source => Generate constructeur using fields...
    2. Décochez le champ reference
    3. Vérifiez que tous les autres champs soient cochés
    4. OK

Maintenant, le constructeur est souligné en rouge car le champ référence n'est pas initialisé. Mais tous les champs ne sont maintenant qu'en jaune. Corrigeons l'erreur sur le constructeur.

5. Ajouter à la fin du constructeur la ligne

    reference = createReference () ;

Maintenant, il n'y a plus d'erreur sur le constructeur mais une nouvelle sur

2 Les champs notés final doivent être initialisés au plus tard dans le constructeur et ne pourront pas être modifiés par la suite.

createReference (). C'est normal, nous n'avons pas défini la méthode. Nous la construirons plus loin. Nous allons d'abord corriger les avertissements sur les champs.

6. Créer les accesseurs et le modifieur pour le prix (hormis le prix, aucun des champs ne sera modifiable ultérieurement)
    1. Menu Source => Generate Getters and Setters
    2. Cochez tous les champs. Cela créera exactement ce que nous attendons
        1. Les champs marqués final n'auront qu'un getter
        2. prix aura getter et setter.
    3. Dans le setter de prix, dans le cas où la valeur du paramètre est négatif, levez une exception `InvalidParameterException` en lui donnant un petit message de type String en paramètre.

Nous allons créer maintenant la méthode qui fait défaut, à savoir createReference (). Le calcul est systématique (trois lettres pour le type d'objet, un numéro de série sur 6 chiffres puis un suffixe spécifique à l'objet. Ce schéma de construction est un patron de conception connu sous le nom de « Patron de méthode ». Un algorithme à trou en quelques sortes.

Pour créer le suffixe, il faudra des informations spécifiques du type de l'objet. Nous commençons par les prendre en paramètre du constructeur sous forme de tableau d'Object[3] avec une syntaxe particulière car nous ne pouvons en déterminer ni le nombre ni le type. Mais ce n'est pas important car on va déléguer la création du suffixe à qui sait le faire au mieux : TypeProduit.

7. Ajout d'un paramètre au constructeur
    1. ajouter en dernier paramètre « complement » au Constructeur de type Object...[4]
    2. ajouter ce paramètre à l'appel de createReference : createReference (complement)
8. On définit maintenant createReference
    1. amenez le curseur de la souris sur createReference et patientez un peu
    2. dans la fenêtre qui apparaît, cliquez sur « create methode createReference(Object[]) »[5]

Nous allons coder cette méthode en utilisant des méthodes que nous définirons plus

3 Toute instance de toute classe est aussi une instance d'une classe particulière nommée Object.

4 Cette syntaxe (Object...) indique que le constructeur ou la méthode va pouvoir prendre un nombre variable de paramètres supplémentaires de type Object. Le paramètre ainsi défini se gère comme un tableau.

5 Le squelette de la méthode est créé à la suite du constructeur.

tard. Donc, il faut s'attendre à des erreurs...

9. Codage de createReference (Patron de méthode)
    1. Supprimez la ligne TODO[6] car nous allons tout de suite écrire le code.
    2. Remplacez return null par :

```
return type.getPrefixe()
      + String.format("%06d", Compteur.newValue())
      + type.getSuffixe(complement);
```

De nouvelles erreurs sur getPrefixe, Compteur et getSuffixe. Normal aucun n'est défini. Une fois de plus elles seront corrigées par la suite. Faites-vous expliquer cette instruction quand vous y serez.

10. Correction de l'erreur sur getPrefixe : nous allons créer cette méthode dans TypeProduit.
    1. Amenez le curseur sur getPrefixe et patientez un peu
    2. lorsque la fenêtre apparaît cliquez sur la solution proposée
        - une méthode getPrefixe () est créée dans TypeProduit.
    3. Cliquez n'importe où dans le fichier pour arrêter l'assistant.
    4. Remplacez le type de la méthode (int) par String
    5. Ajoutez le mot clé abstract après public (ceci générera de nouvelles erreurs).
    6. Remplacez le corps de la méthode par un simple ; .
    7. Amenez le curseur sur la première erreur et cliquez sur « fix 5 problems ... »
    8. Regardez ce que vous avez obtenu et faites vous expliquer ce que vous venez de faire.
    9. Dans chacune des méthodes getPrefix
        1. supprimez la ligne TODO
        2. remplacez null par la valeur appropriée.
11. Faites de même pour getSuffixe
    1. retournez à Produit
    2. faites les étapes 1, 2, 3, 5 et 6 (la 4 n'est pas nécessaire car le type de getSuffixe est correct, et nous ne créerons les variantes qu'après avoir créé les types spécifiques).
12. Création de la classe Compteur

6 Eclipse insère souvent une ligne de ce genre quand il créé automatiquement certains éléments notamment des constructeurs ou des méthode alors qu'il ne sait pas créer de code correct. Il s'agit d'un marqueur pour vous rappeler que vous aurez à écrire le code correct à cet endroit.

1. retournez à produit
2. amenez le curseur sur Compteur
3. choisissez la première option (create class Compteur)
    1. dans le champ package mettez utilitaires pour structurer proprement vos sources[7]
    2. Finish (le reste du formulaire convient)
4. mettez ce qui suit dans le corps de la classe et faites-vous expliquer

```
private static Compteur instance = null;

private int suivant = 1;

private Compteur () {
}

public static int newValue () {
      if (instance == null) {
            instance = new Compteur ();
      }
      return instance.increment ();
}

private int increment() {
      return suivant++;
}
```

La suite sera publiée la semaine prochaine.

7 Votre fil rouge comportera un bon nombre de classes lorsqu'il sera fini. Autant ranger les classes tout de suite.