CNAM

M-NFA032 Algorithmique et programmation : bibliothèques et patterns

Sixième 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.

Thème de la semaine

Cette semaine, nous allons nous consacrer à la sauvegarde et restitution des données. Les classes qui nécessitent une sauvegarde et une restitution sont :

- Compteur :
  il faudra sauvegarder et restituer l'état du compteur de manière à ne pas réutiliser les nombres générés ;
- Catalogue :
  il faudra sauvegarder, le contenu de l'instance et être capable de le restituer ;
- CarnetClientele :
  comme pour Catalogue, il faut être capable de sauvegarder et restituer le contenu de l'instance ;
- CarnetCommandeMagasin :
  de nouveau comme Catalogue.

Les autres classes seront sauvegardées automatiquement grâce au mécanisme de sauvegarde d'objets.

On mettra enfin en œuvre un mécanisme pour demander à l'utilisateur si il souhaite faire une sauvegarde lors de l'arrêt du programme si des modifications ont été apportées à des données sauvegardées.

Dans les quatre classes, nous allons créer une méthode de sauvegarde et une méthode de restitution. Ces méthodes prendront en paramètre un ObjectOutputStream ou un ObjectInputStream ouvert qui sera le flux dans lequel il faudra sauvegarder ou restituer les données.

Pour prévoir d'éventuelles évolutions de l'application qui peuvent générer de nouvelles données à sauvegarder, nous allons créer un gestionnaire de sauvegarde.

Dans un premier temps, nous définirons une interface pour décrire le comportement de ce qui est à sauvegarder. Dans un second temps, nous définirons le gestionnaire et ses services : ajout d'une entité, sauvegarde, restitution, … Enfin, nous adapterons les 4 classes à sauvegarder.

Exercice 1 : Création d'une interface

Pour que la suite fonctionne bien, les classes implémentant cette interface doivent toutes implémenter le patron singleton et, donc, fournir un service statique public static xxx getInstance () qui permettra de récupérer l'instance. Ceci n'est pas exprimable directement en Java, mais la documentation de l'interface devra le mentionner.

1. Dans le package utilitaires, créer une interface nommée « Sauvegardable ».
2. Définir une **procédure** nommée « enregistrer » et prenant en paramètre un ObjectOutputStream. Le rôle de cette méthode est d'effectuer la sauvegarde dans le flux, des données d'une classe implémentant cette interface. Cette procédure est susceptible de lever soit une IOException. Pensez à déclarer que cette m éthode est susceptible de lever cette exception.
3. Définir une **procédure** nommée « restituer » et prenant en paramètre un ObjectInputStream. Le rôle de cette méthode est d'effectuer la restitution des données depuis le flux pris en paramètre. Cette procédure est susceptible de lever une IOException et une ClasseNotFoundException. Pensez à déclarer ces exceptions comme étant susceptibles d'être levées.

Exercice 2 : Création du gestionnaire de sauvegarde

Nous allons avoir besoin d'utiliser la réflexion de Java permettant l'introspection des classes, c'est-à-dire de charger une classe, d'en créer une instance et d'accéder aux membres de la classe sans connaître la classe par avance. Le code nécessaire et commenté de ce mécanisme vous est fourni. Vous n'êtes pas obligé de comprendre en détail le mécanisme pour vous en servir. C'est hors sujet pour cette UE. Par contre, vous aurez d'autre code à ajouter...

Singleton

1. Créer une classe nommée « GestionnaireSauvegarde » dans le package utilitaires.
2. Appliquer le patron singleton à cette classe.

Gestion des classes

1. Copiez-collez la définition de la variable d'instance ci-dessous dans la classe.

```
/**
 * Cette variable d'instance permet de collecter toutes les classes à
 * sauvegarder. Il suffit d'ajouter à cette liste chacune des classes à
 * sauvegarder et à restituer dans le constructeur
 *
 * @see GestionnaireSauvegarde#GestionnaireSauvegarde()
 */
private LinkedList<Class<? extends Sauvegardable>> aSauver;
```

2. Ajoutez les lignes suivantes dans le constructeur privé de « GestionnaireSauvegarde ».

```
            // création de la liste
            aSauver = new LinkedList<>();
            // ajout des classes à sauvegarder à la liste aSauver. Si de
nouvelles
            // classes sont à ajouter, il n'y a qu'à les mettre après
            // CarnetCommandeMagasin.class.
            // Compteur.class est un objet de type Class décrivant le contenu de
la
            // classe Compteur. C'est une syntaxe classique de Java. Il en va de
            // même pour les trois autres classes.
```

```
        Collections.addAll(aSauver, Compteur.class, Catalogue.class,
                    CarnetClientele.class, CarnetCommandeMagasin.class);
```

Il devrait y avoir une erreur sur addAll qui disparaîtra d'ici la fin de cette feuille.

3.
4. Ajouter la méthode suivante à la fin du corps de GestionnaireSauvegarde.

```
    /**
     * permet de récupérer l'unique instance de la classe dont le descripteur
     * est donné par le paramètre {@code clazz}. La classe doit offrir une
     * méthode {@code getInstance} statique et sans paramètre.
     *
     * @param clazz
     *            la classe dont on veut une instance.
     * @return l'unique instance de la classe.
     * @throws IllegalAccessException
     *             si les droits d'accès à getInstance ne sont pas suffisants.
     * @throws IllegalArgumentException
     *             si les paramètres de {@code getInstance} ne conviennent
pas.
     *             Ceci ne devrait jamais se produire.
     * @throws InvocationTargetException
     *             si la méthode getInstance lève une exception particulière.
     * @throws NoSuchMethodException
     *             si la classe ne définit de méthode getInstance()
     * @throws SecurityException
     *             si il y a un problème d'accès au package de la classe lié à
     *             la sécurité.
     * @see Sauvegardable
     */
    private Sauvegardable getInstance(
            Class<? extends Sauvegardable> clazz)
                    throws IllegalAccessException,
                    IllegalArgumentException,
InvocationTargetException,
                    NoSuchMethodException, SecurityException {
        return (Sauvegardable) clazz.getMethod("getInstance")
                .invoke(null);
    }
```

Données modifiées

1. Ajoutez à la classe une variable d'instance booléenne « modifie » qui servira à se rappeler si des données ont été ajoutées, supprimer ou modifiées dans la base de données.
2. Initialisez ce champ à false dans le constructeur.
3. Ajoutez une méthode « marquer » qui permette de le forcer « modifie » à true. Cette méthode sera utilisée pour signifier au gestionnaire qu'une donnée sauvegardée a été modifiée.
4. Ajoutez un accesseur à ce booléen.

Sauvegarde

1. Définissez dans « GestionnaireSauvergarde », une méthode sauver prenant en paramètre un descripteur de fichier de classe java.io.File.
2. Commencez a mettre dans cette méthode un squelette permettant d'ouvrir un ObjectOuputStream nommé « oos ». N'oubliez pas de vous assurer que les ressources mobilisées par l'ouverture du fichier soient libérées dans toutes les circonstances, sinon, vous allez saturer la mémoire de la JVM ! N'essayez pas de capturer ici les IOExceptions que les créations d'instance peuvent lever car nous ne savons pas quoi en faire ici. Déclarez simplement que cette méthode est susceptible de les générer. Par

contre, filtrez celles qui peuvent arriver lors des fermetures comme montré en cours.

3. Ajoutez les lignes suivantes à la fin du corps du try. Ces lignes permettent d'appeler la méthode « enregistrer » de toutes les classes mises dans « aSauver ».

```
for (Class<? extends Sauvegardable> clazz : aSauver) {
      getInstance(clazz).enregistrer(oos);
}
```

4. Ajoutez, en utilisant le « quick fix » d'Eclipse, les différentes exceptions que ces dernières lignes peuvent générer à la clause throws de la méthode. De nouveau nous ne pouvons pas les gérer ici.
5. À la fin de la méthode, si tout c'est bien passé, forcez le champ modifie à false.
6. Créez finalement une surcharge de la méthode sauver prenant en paramètre le chemin du fichier de sauvegarde. Elle appelle uniquement la méthode précédente en créant une nouvelle instance de File avec, en paramètre, le chemin d'accès reçu en paramètre.

Restitution

1. En procédant de la même manière que pour la sauvegarde, créez un service « restituer ». Dans la boucle for donnée ci-dessus, il faudra remplacer enregistrer par restituer bien entendu.

Exercice 3 : Implémentation de l'interface

Il ne reste plus qu'à implémenter l'interface dans les quatre classes à sauvegarder.

Compteur

1. Il faut apporter quelques modifications à la classe Compteur pour répondre au contrat de Sauvegardable. Remplacez la méthode newValue () qui vous a été donnée en atelier par :

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

public static int newValue() {
      return getInstance().increment();
}
```

2. Dans la méthode « increment », avant le return, ajoutez signalez au gestionnaire de sauvegarde que le compteur a été modifié en ajoutant la ligne :

```
GestionnaireSauvegarde.getInstance().marquer();
```

3. Ajoutez l'interface « Sauvegardable » dans la clause implements de la classe.
4. À l'aide du « quick fix » d'Eclipse,
   1. ajoutez les méthodes exigées par l'interface puis définissez le corps de ces méthodes. Un simple ordre de lecture ou d'écriture du champ suivant est suffisant (Cf. cours).
   2. Laissez passez les exceptions potentielles.

CarnetClientele

1. Ouvrir CarnetClientele
2. Ajouter Sauvegardable dans la liste des interfaces implémentées. Une erreur doit apparaître sur CarnetClientele vous signalant que cette classe doit implémenter les méthodes héritées : sauvegarder et restituer
3. Avec le « quick Fix » choisissez « Add unimplemented methods »
4. Aller sur la méthode enregistrer qui vient d'être créée
5. Nous allons nous contenter d'enregistrer la HashMap englobée dans la classe. Pour ce faire, remplacer le corps de enregistrer par la ligne suivante :
   ```
   oos.writeObject(contenu);
   ```
   où contenu est à remplacer par le nom de la variable d'instance typée HashMap<String,Client> que vous avez choisie.
6. Ensuite, aller sur restituer
7. remplacer le corps de la méthode par la ligne
   ```
   contenu = ois.readObject();
   ```
   où, de nouveau, contenu est à remplacer par le nom de la variable d'instance.
8. Une erreur doit apparaître sur ois.readObject. Pour la corriger, choisir « Add cast ... » dans le « quick Fix ». Cela ajoute (HashMap<String><Client>) devant pour signifier que l'objet lu est à convertir en HashMap<String><Client>
9. Il reste un avertissement sur la ligne. Pour le faire disparaître, choisir « Add @SuppressWarnings ... » dans le « quick fix »

Catalogue

1. Faire les mêmes opérations pour Catalogue en adaptant à la structure de données, bien entendu.

CarnetCommandeMagasin

1. Pour CarnetCommandeMagasin, nous allons déléguer à CarnetCommande.
   1. Procéder comme pour CarnetClientele et Catalogue pour la classe CarnetCommande.
   2. Dans la classe CarnetCommandeMagasin, Faire les points 1, 2 et 3 expliqués ci-dessus pour CarnetClientèle.
   3. Dans la méthode enregistrer, appeler simplement la méthode enregistrer sur la variable d'instance de type CarnetCommande. C'est une délégation.
   4. Faire de la même sorte pour restituer.

Normalement, l'erreur dans GestionnaireSauvegarde doit avoir disparue. C'est normal, la ligne sur laquelle il y avait l'erreur attendait que toutes les classes présentes implémentent Sauvegardable. Jusqu'à présent, toutes ne le faisaient pas, mais maintenant, ce n'est plus le cas.

Exercice 4 : Signaler toute modification au gestionnaire de sauvegarde

Il faut maintenant prévoir de signaler au gestionnaire de sauvegarde, toute modification apportée aux différentes données. Nous les passons en revue ci-dessous. Il suffira d'ajouter à la fin de la méthode la ligne de l'étape 2 de Compteur ci-dessus.

CarnetCommande

- Méthode d'ajout d'une commande.

Commande

- Méthode qui verrouille la commande.

Client

Rappels : le panier n'est pas sauvegardé ni restitué et le carnet de commande gère lui-même les ajouts de commandes.

- Les différents modifieurs des champs.

Produit

- La modification du prix.

Lot

- Bien que pas strictement obligatoire ici, l'ajout d'une quantité de produit au lot.

Normalement ceci suffit. Ce sont les seules modifications que l'on peut apporter aux données et qui demandent une sauvegarde pour ne pas perdre les données.

Exercice 5 : Vérification

Vous pouvez créer un main de test qui créera quelques données (des produits, des clients et des commandes). Pas besoin d'une grande quantité. Ensuite, le programme enregistrera les données dans un fichier (GestionnaireSauvegarde.getInstance().sauver(….)). Puis, il relira les données depuis le fichier. Enfin, il vérifiera que tout a été restitué correctement.