Logiciel de Gestion d'Entrepot
- Rapport -

Tristan Savaria et Jonathan Forget

9 décembre 2013

Table des matières

1 Description et utilisation des fonctions

1.1 Utilisateur

Le programme est un gestionnaire d'entrepot pour une compagne de vente à grande surface. Le programme importe un fichier .txt d'encodage UTF-8 et de format CSV au lancement pour initialiser les produits en entrepot, les clients et les commandes. Le programme de gestion supporte deux types de produits, soit les livres et les ordinateurs. Au dessus de l'interface ce trouve deux menus. Celui de gauche pour sélectionner le type de produit et celui à sa droite pour sélectionner une action : Ajouter, Enlever et Commander (l'action agit sur le type de produit). La sélection "Ajouter" permet de rajouter un nouveau produit dans l'entrepot, une nouvelle ordinateur par example. Pour ce faire, il faut correctement écrire dans les champs décrivant le produit situé en dessous. Enlever retire le produit en entier de l'entrepot, cependant il figurera toujours parmis les commandes ultérieures des clients. Commander permet de commande un produit en entrepot à un de vos clients, la quantité commandé sera automatiquement retiré de la quantité en entrepot. Pour confirmer chacune des actions, le bouton "OK" situé au dessus de l'interface doit être cliqué. Le bouton "Lister" permet d'afficher tout les produits en entrepot, les commandes et les détails des clients. La configuration des deux menus n'ont aucune incidence sur l'affichage. Lorsque le programme est fermé, le fichier de l'entrepot est écrasé avec la nouvelle version figurant vos modifications.

1.2 Programmeur

La classe *LecteurEntrepot* est responsable de l'importation et la sauvegarde du fichier d'entrepot d'encodage UTF-8 et de format CSV. Le type d'objet supporté doit être manuellement rajouté dans la liste nommé "objetSupporte", l'opération est trouvé à l'intérieur du constructeur de base. La méthode LecteurEntrepot, à partir du chemin du fichier passé en paramètre, extrait chaque ligne du fichier et à l'aide de la variable boolean "isInitialized", indique si un problème de lecture a eu lieu. La méthode TraitementLigne, prend chaque ligne passé en paramètre et, selon le format CSV, extrait les champs et les retournent dans une ArrayList de String. La méthode Instantiation prend une List de String, compare le premier champs (le type d'objet) avec le contenue de objetSupporte pour trouver un type adéquat. La méthode vérifie et signale les duplications, si aucune duplication existe et que le type est adéquat, un objet est instantié et enregistré dans la collection approprié. Les produits sont enregistrés dans un TreeMap<String, Produit> où la clée est le code du produit, les clients sont enregistrés dans un TreeMap<String, Client> où la clée est le nom du client et les commandes sont enregistrées dans un ArrayList. La méthode Sauvegarde prend en paramètre le chemin vers un fichier pour l'écriture et les trois mêmes type de collections détaillées ci-dessus pour prendre l'information et le transformer en format CSV sous un fichier sur le disque.

La classe *Livre* hérite de la classe Produit. Livre a un champs private String nommé auteur et un champs private String nommé titre. Pour chacun de ces champs, un getter publique est disponible pour avoir accèss à l'information. La classe surcharge la méthode "toString()" de la classe mère pour pouvoir retourner le champs auteur et le champs titre.

Ordinateur hérite de la classe Produit. Ordinateur a un champs private String marque et un champs private int capaciteStockage. capaciteStockage représente la mémoire disque du produit en question. La classe surcharge la méthode "toString" de la classe mère pour pouvoir retourner le champs marque et le champs capaciteStockage

La classe *JTextAreaOutputStream* est une solution de Mikhail Vladimirov (http ://stackoverflow.com/que
out-println-to-jtextarea) pour rediriger le flue de la console vers un JTextArea. La classe hérite de OutputStream et a un objet private final JTextArea où le texte sera affiché. De l'héritage, la méthode "write" est surchargé pour prendre "text" et opporé un append sur le JTextArea pour rajouter le texte.

La classe *Entrepot* est la classe métier du projet, elle est responsable des opérations de l'utilisateur. Cette classe instantie un objet LecteurEntrepot ainsi que les collections nécessaire à l'enregistrement des commandes, des clients et des produits (voir classe "LecteurEntrepot" pour description des types). La méthode ChargementDonne prend en paramètre le chemin vers le fichier à charger et lance la méthode LectureFichier de l'objet lecteurDonne. Ensuite, elle vérifie si le lecteur a rencontré une erreur et si ce n'est pas le cas, les collections enregistrées dans le lecteur seront à leur tour enregistré dans l'objet Entrepot. La méthode RajoutUnite est en dépréciation, elle prend le code d'un produit et un nombre de quantité, vérifie la quantité, vérifie que produits contient bien une valeure associée au code du produit et ajoute la quantité disponible au produit. La méthode EnleverUnite est également en dépréciation et fait l'opération contraire de RajoutUnite. La méthode Commander prend en paramètre le code du produit, la quantité, le nom et l'adresse du client. Si le code du produit fait déjà partis de produits, la quantité disponible est vérifié avant d'enregistré la commande au client. Si le client est un nouveau client, il est rajouté dans clients. Les méthodes ListerProduits, ListerCommandes et ListerClients utilise System.out pour imprimé dans un tableau toutes les informations contenues dans les objets clients, produits et commandes. RetirerProduit prend le code d'un produit et le retire du TreeMap produits. NouveauLivre et NouveauOrdi instantie un nouveau objet du même type et le rajoute dans produits. Finalement, Enregistrer lance la sauvegarde de lecteurDonne en donnant en paramètres le chemin du dernier fichier lut, produits, clients et commandes.

La classe *EntrepotGui* est un JFrame et est responsable de la partie interface du programme. Pour effectuer les opérations, la classe a besoin d'un objet Entrepot passé en paramètre dans le constructeur. La méthode processWindowEvent (solution trouvé à http ://stackoverflow.com/questions/15499211/calling-function-on-windows-close) permet de lancer la méthode Enregistrer de l'entrepot lors de la fermeture du programme.

Le GUI est un arrangement de combosBox, d'un CardLayout, etc. Dans la méthode Initialisation, le JTextAreaOutputStream est initialisé avec le JTextArea. Également, la redirection de System.out a lieu dans cette méthode.

2 Structures de données employées et justification

Aucune structure de données n'a été utilisé.

3 Discussion

La difficulté du projet résidait dans la création de l'interface de l'utilisateur. Nous avons eu de la difficulté a organisé le code et parfois l'emplacement des composantes n'était pas celle voulu. Pour les produits, nous avons décidé de choisir un TreeMap<String, Produit> avec le code du produit comme clée. Avec ce type de collection nous pouvons facilement accéder à un produit avec son code,TreeMap permet d'ordonner les produits avec les codes automatiquements. Par example, si nous le client du logiciel enforce une politique que chaque code de produit du même type débute avec la même lettre, tout sera ordonnée automatiquement. Pour les clients, la collection choisie est un TreeMap<String, Client> où la clée est le nom du client. Stipulé dans la classe Client, le seul attribut comparé pour différentier un client d'un autre est le nom. Il est donc trivial de choisir le nom comme clée. En conséquence, un deuxième client avec le même nom entrera en conflit avec le premier, mais puisque la classe Client ne peux pas distinguer entre les deux de toute manière, il est donc acceptable d'utiliser le nom comme clée dans la map. Dans les deux collections, une liste n'est pas idéal, car il faut soit connaitre la position de l'objet dans la liste pour retourner l'objet ou avoir une copie de l'objet lui même. Nous ne pouvons donc pas élaborer des opérations à partir d'informations partiels sur l'objet comme le code ou le nom. Une map détient seulement une valeure pour une clée, elle empêche donc la duplication tout comme les Sets. Les commandes sont enregistré dans un ArrayList. Il n'y avait aucune justification pour prendre une Map puisqu'il n'est pas requis de faire une recherche dans les commandes avec un champs quelconque. Aussi, il était moins évident qu'est-ce qui agirait comme clée. Un Set n'a pas été choisis. Nous pouvons penser qu'un Set serait idéal, car il empêche un client de passer exactement la même commande plus d'une fois. Cependant, le format de Date tel que définit dans la classe Commande est précis à la minute près. Il est donc impossible de distinguer entre deux commandes passé à quelques millisecondes près. D'ailleur, les clients ce distingue par leur nom. Par l'utilisation des Sets il aurait été possible que deux clients appelé "Guy" commande le même article, la même quantité, à quelque millisecondes près sans pouvoir distinguer les deux commandes ! Nous avons donc préférer mitiger cette possibilité d'erreur avec un ArrayList.

4 Améliorations possibles

Pour le code, la classe EntrepotGUI est mal structuré et difficile à si trouver. Dans LecteurEntrepot, nous avons utilisé une petite astuce pour reconnaitre le type d'objet supporté, mais un client du logiciel devra modifier manuellement le code de cette classe à quelques endroit pour rajouter un nouveau type. Pour le fonctionnel, nous pourrions rajouter des JComboBox pour que le bouton Lister puisse faire le listage du type de produit voulut, des commandes ou des clients plutôt que faire le listing de tout au complet. Finalement, lors du rajout d'un produit, le GUI ne valide pas le type des champs, une exception sera donc lancé au lieu d'afficher un simple message d'erreur.