* l'évolution de la modélisation

Après le premier TP, j’ai fait un modèle primitif avec DP Observer pour l’affichage du dernier élément produit. L’image de ce modèle UML.png se trouve dans le répertoire. Par rapport au modèle UML final, je remplace les deux classes Publisher/Subscriber par deux interfaces, car je trouve qu’il est un peu ambigu de mettre les attributs de Pile dans le class Publisher. Donc maintenant la classe Pile réalise l’interface Publisher et les classes qui ont besoin peut réaliser l’interface Subscriber. Par ailleurs, je utilise ArrayList pour gérer la liste de Subscribers. C’est plus facile.

Je pensais que la classe Log réalise aussi l’interface Subscriber. Mais pour chaque pile, il faut avoir son propre fichier log. Dons j’ajoute un attribut PILLOG dans la classe Pile. Le nom du fichier log est donné quand on crée un objet Pile.

Pour les alarmes, j’ai créé une classe mère et des classes fils qui réalisent l’interface Subscriber. Ils n’ont pas beaucoup changé.

Pour la classe Stock, elle réalise aussi l’interface Subscriber. Avec plus de consignes, j’ajoute une fonction pour l’affichage du dernier élément produit et une autre pour l’affichage de tout le contenu du stock.

Ensuite, quand je commence à programmer, j’ajoute une classe Command pour recevoir les instructions saisies depuis le clavier.

Pour la gestion d’alarmes, j’applique DP Strategy. Donc il y a une interface Warn qui gère les différentes façons d’alarmes qui correspondent à la saisie SET WARN 1, SET WARN 2. Donc, j‘ajoute un attribut PILWarn dans la classe Pile qui permet de changer dynamiquement de mécanisme de la gestion d’alarmes. Je pense que les alarmes précédentes sont similaires à Warn, donc j’ajoute une interface Alert. Les deux méthodes d’alarmes la réalisent.

Pour la classe Warn1, elle analyse chaque modification du stock pour définir s’il y a besoin d’alerter ou non en fonction de nombre d’éléments restants. Donc il y a une réalisation de l’interface Subscriber. Pour Warn2, elle compte le nombre d’éléments par deux secondes.

Pour finir, j’ajoute deux classes Producer et Consumer pour créer des threads producteur et consommateur. Le producteur produit un entier toutes les 20s et le consommateur consomme un entier toutes les 30s. Il faut savoir dans quel pile il produit/consomme donc il y a une agrégation entre la classe Pile et les deux.

* la justification des choix et les limites

1° Dans quelles classes devez-vous intervenir (modifier une instruction) si on souhaite supprimer un objet qui « surveille » votre stock (par ex. supprimer une alarme, la construction du log., un consommateur,…) ?

Pour supprimer une alarme, on peut ajouter une ligne PIL.PILRemoveSubscriber(...) dans la classe main. Pareil pour supprimer un consommateur.

Pour modifier la construction du log, on peut ajouter une ligne PIL.PILLogWrite(...) dans la classe main.

Vous changez seulement la partie de code qui s’occupe de l’applicatif ou vous changez d’autres méthodes, classes ?

Il faut changer seulement la partie de code qui s’occupe de l’applicatif.

Devez-vous modifier votre classe de stockage ?

Non.

2° Même question si vous devez rajouter un objet qui affiche tout le contenu du stock.

J’ajoute une fonction dans la classe Stock qui permet d’afficher tout le contenu du stock.

3° Est-ce que les méthodes/classes s’occupant des alarmes sont sollicitées bien plus souvent que nécessaire ?

Oui, lors de chaque modification du stock.

* Egalement n'hésitez pas expliquer votre modélisation (DP utilisés, de quelle façon, adaptations ...)

DP Observer

Publisher : Pile

Subscriber : Stock, Alert1(Alert1Empty, Alert1AlmostEmpty, Alert1AlmostFull, Alert1Full), Warn1.

DP Strategy pour gérer dynamiquement la façon d’alarmes. La classe Warn1 réalise l’interface Warn et à la fois l’interface Subscriber.

Warn1 : afficher une alarme quand le nombre d’éléments est supérieur à 7.

Warn2 : afficher une alarme s’il y a plus de 2 éléments produits ou consommés dans les deux secondes.

* la ligne de commande à utiliser pour exécuter votre programme sachant que je ferai un copié/collé dans la console à partir du répertoire où se trouve le jar

java –jar Stock.jar

(PUSH ..., POP, Clear, SET WARN 1, SET WARN 2, q)