TP sur la State Machine

1 Contenu du TP

- Le but de ce deuxième TP (sur les fondamentaux de la conception Objet en Java), est de vous présenter un cas d'utilisation pratique du *Design Pattern State* (Famille de *Comportement*) et de sa *Machine d'Etat* associée.

- Ce modèle de composant est TRES largement/couramment utilisé dans les projets de développement contemporain de l'Industrie.

  Il fait parti des très grands classiques du métier de Développeur, et bénéficie d'ailleurs d'une normalisation/reconnaissance à travers de nombreuses technologies (*Diagrammes d'Etats UML*, *Frameworks*, outils, …) ce qui démontre aisément le large spectre technique qu'il couvre.

  Ce pattern est la transposition du principe « *d'états* associés à des *objets* », son universalisme dans bon nombre de matière est reconnu (Marketing, Finance, Biologie, Mathématique, Physique, Sciences Sociales, etc…).

- Le code source de ce projet (à importer dans Eclipse) repart du TP précédent sur la conception Objet en Java, et y greffe un nombre important d'évolutions…

- Le *Model* détient 2 nouveaux sous packages :

  - **state** : Il présente une *Enumération (EOrderState),* déclarant de façon Déterministe les divers Etats, et leurs références (*TRANSACTION_NULL*, …) ; ainsi que leurs séquences d'Enchaînement (chaque état pointe sur son suivant…).

    Une interface *IStateChangeable*, héritant de *IValidable* : le comportement/contrat de « changement d'état » est une extension d'une fonctionnalité de *Validation* (chaque transition est donc ainsi sujette à *validation*, avant *modification*).

    L'*OrderStateMachine*, se charge d'implémenter concrètement la machine d'état de notre projet, en validant d'abord l'intégrité / la cohérence des données, avant d'en modifier l'état.

  - **validation** : Un package simple, présentant *l'Interface Mère* (dans la hiérarchie objet/de type) *IValidable*, et une classe *d'Exception*, à lancer en cas d'échec lors de la validation d'un objet du modèle.

- Une modification simple a été apportée sur le modèle :

    - La classe *Order*, (structurant une « commande client »), déclare l'implémentation de l'interface *IStateChangeable*.

      Puisque l'on « enrôle » cette classe dans la *StateMachine*, elle se doit d'adopter ce comportement (la *StateMachine* n'accepte/ne fonctionne que sur des types abstraits issus de cette Interface, cf *OrderStateMachine.changeStep(IStateChangeable ref)*)

      NB : On remarquera l'élégance de l'approche *Orienté Objet* des *Interfaces* en Java…

- De même nous avons modifié la Classe *Main* pour y compléter notre exemple :

    - A la fin de la méthode *main*(…), la *StateMachine* est appelée pour opérer la modification de l'état d'une commande utilisateur.

    - La première fois, l'ordre o, de type *Order* (mais surtout *IStateChangeable)* est invalidé car une de ses références (de son arbre d'objets) est absente (la propriété « client »), la Machine d'Etat lance donc une exception (qui est *try-catchée* selon les règles usuelles), le changement n'est pas enclenché.

    - La méthode *main* ensuite corrige le problème en affectant la référence « client », et ainsi, le second appel à la Machine ne présentant pas d'incohérence dans les données présentées, la modification peut avoir lieu.

    - NB : ici SEULE la *StateMachine* est responsable de la bascule d'état, et sur des objets qui sont définis pour un seul Type, *IStateChangeable*, issu de son propre package (pas de conflits sur des dépendances extérieures)

      La responsabilité de la *Classe Order* du *model*, est de porter la définition concrète de la validation d'un objet (de par la méthode *validate()*) dont il a la charge).

      Cette segmentation des rôles propre à chacun (Une machine ne valide que des objets typés par son interface ; le modèle doit définir les règles métiers dont il veut être le représentant), offre un exemple de conception générique et réutilisable.