TD-TP2

L'application `Order`

Cette application permet la constitution d'une commande. On peut commander des articles (Article), des articles en promotion (DiscountedArticle) ou des lots (Pack).

Le code de l'application est donné .

Exemple d'exécution de la classe OrderTester

```
        O R D E R


Pack: Hammer - Assorted nails - : $29,90
discounted pen: $0,60
discounted ruler: $1,70


TOTAL DUE: $32,20
```

Question 1

1.1) Faire la rétro-conception du code avec Visual-Paradigm pour obtenir le diagramme de classes de conception. N'oublier pas d'ajouter les dépendances et d'améliorer le diagramme notamment pour les réalisations d'interface.
1.2) Un patron de conception est utilisé dans une partie de l'application ? Lequel ?

1.3) Expliquer le rôle joué par les classes dans le patron en tant que participant (par rapport à la définition du patron). Vous pouvez annoter le diagramme de classes pour indiquer les rôles.

Question 2 : Lire l'ensemble de la question avant de répondre

Maintenant on remarque que notre application n'est pas encore complètement terminée.

En effet un attribut nommé « status » permet d'indiquer dans quel état est la commande :

- au départ elle sera en cours (Pending), status = 0
- puis elle sera validée (Confirmed) , status = 1
- et à la fin en livraison (Delivered), status =2

Question 2-1 : Enumération

Modifier la classe `Order` pour remplacer les valeurs numériques de l'attribut `status` par une énumération en utilisant les libellés ci-dessus. Vous pouvez appeler votre énumération `State`.
Ne pas oublier UNDETERMINED.
Donner le code Java de l'énumération et les modifications à apporter dans la classe `Order`.

Question 2-2 : Exception

Donner le code des méthodes `nextState()` et `setStatus(State s)` après avoir :

- remplacé dans la méthode `nextState()` le `switch` par des `if` imbriqués et lancer une exception dans le cas indéterminé. Pour cela définir une classe d'exception `OrderException` ;
- ajouté une méthode `setStatus(State s)` qui permet de changer l'état de la commande directement.

Question 2-3 : le patron State

Quand une commande est en cours on peut ajouter ou retirer des produits, par contre quand elle est validée on ne peut plus ajouter de produits mais on peut en retirer, et bien sûr quand elle est en livraison on ne peut plus faire de modification.
Les changements d'états sont provoqués par l'utilisateur qui demandera explicitement à une commande de passer à l 'état suivant en lui envoyant le message « nextState() ».
On voit que pour l'instant notre application ne gère pas les contraintes liées au statut de la commande. Nous allons remédier à ce problème et simplifier le code en appliquant le patron State.

- 1) Expliquer l'utilisation du **patron State** dans cet exemple (objectif du patron et rôles des classes participantes)
- 2) Dessiner le diagramme de classes montrant l'introduction du patron State pour une commande en ne reprenant du diagramme de classes de l'énoncé que la classe `Order`. Ne pas oublier de mettre toutes les méthodes nécessaires et d'assurer les liaisons entre la commande et ses états et vice versa.
- 3) Maintenant que la classe `Order` doit déléguer une partie de son travail à ses états :
  - modifier le code de la classe `Order` afin que le patron State fonctionne (attributs, modifications des méthodes, …) et donner le code java de la nouvelle classe `Order`.

  Attention il ne doit plus rester de conditionnelles.

- 4) Les états ont besoin également d'une méthode pour réaliser les changements d'états. Ajouter le code Java de la méthode `nextState()` pour chacune des classes d'état.

- 6) Refaire un scénario en modifiant la méthode `main` de la classe `OrderTester` (c'est l'utilisateur qui provoque les changements d'états) et faire les modification dans le code pour que tout fonctionne.