TD N°3

Les limites de l'Héritage, *Design Pattern Decorator* et *Builder*

Nous poursuivons dans cet exercice ce qui a été fait la dernière fois en observant que l'héritage trouve ici ses limites. En effet, une classe de produits qui serait à la fois alimentaires, vendus au volume, mais par lots, auquel on applique une propotion pour certains clients, etc. serait une classe héritant de nombreuses autres classes dans la logique de l'héritage.

Comme nous n'utilisons pas l'héritage multiple, qui est difficile à mettre en place et impossible en Java, nous allons proposer de mettre en place le *Design Pattern Decorator*.

1. Définissez l'interface `Product` où un produit est défini par les méthodes :
   - `double getPriceExcludingVAT()`
   - `double getPriceIncludingVAT()`
   - `double getVATAmount()`
2. Créer la classe `CostumerProduct` qui implémente `Product`
   - name, le nom de l'article sous la forme d'un `java.lang.String`. Ce nom doit être unique dans les collections de produits.
   - price, Le prix en euros hors taxes enregistré sous la forme d'un double
   - VAT, Le montant de la taxe (20, 10, 5.5 ou 2,1 %) sous forme d'une constante de type double
   - id de type `java.util.UUID` qui sera l'identifiant permettant de retrouver un produit dans les collections de produits

     On implémentera la méthode virtuelle `double getVAT()` pour une taxe de 20%
3. Créer une classe abstraite `ProductDecorator` qui implémente `Product` et qui délègue à `private Product decoratedProduct` un produit dont on ajoutera des propriétés dans chaque `Decorator`.
4. Créer un nombre important de `Decorator` pour couvrir différents besoins en imaginant les situations possibles :
   - vendu au volume
   - vendu au poids
   - vendu en lots
   - avec date limite de consommation (interface `Deadline`)
   - vendu avec les TVA 2.1, 5,5, 10 et 20%
   - produits dangereux (inflammables, explosifs, toxiques, armes)
   - avec remise pour certains clients
   - qui donne une offre sur carte de fidélité
   - avec remise sur d'autres produits
   - etc.
5. Créer l'objet principal, y définir des produits complexes comme par exemple unp roduit alimentaire vendu au litre, par bouteilles de 50cl, en lots de 12, dans des cartons de 10, avec une date limite de vente fixée au 1er janvier 2026.

   (a) Proposer un mécanisme pour détecter un *decorator* appliqué à un produit et appliquer cette information sur un autre textitdecorator. Par exemple, appliquer une propotion d'un produit offert pour toute vente en lot de 12.

   (b) Quel constat faisons-nous ?
6. Pour construire des produits en appliquant des *Decorator*, appliquer le *Design Pattern Builder*
   - Écrire la classe `ProductBuilder`

— Y développer la méthode `Product build()` qui renvoi une instance de produit
— Y développer des méthodes `ProductBuilder setQuantity(double quantity)`, `ProductBuilder se`
etc. pour appliquer à ce produit des propriétés.
— Y développer des méthodes pour prendre en compte les interactions entre différentes propriétés (par exemple l'incompatibilité de formes de vente, l'application de propotions à certains types de produits, etc.).