Travaux Dirigés 2 – Du bon usage de l'héritage

Dans cet ensemble d'exercices, nous refaisons un nouveau projet avec les mêmes concepts de la feuille précédente, mais sans reprendre nécessairement les mêmes classes (vous pouvez dupliquez votre projet `td1` dans un projet `td2`).

1. **Héritage**

   La classe `ConsumerProduct` représente les produits commercialisés. Ils ont un nom, un prix hors taxes, la taxe TVA appliquée qui est de 20% par défaut.

   Nous voulons ajouter des produits alimentaires. Ceux-ci sont taxés à 5,5%. De plus, la vente des produits alimentaires est réglementée et chaque produit possède une date limite de vente (DLV).

   (a) Construire l'ensemble des classes et interfaces `Product`, `ConsumerProduct`, `FoodProduct`, `UnitProduct`, `FoodProductSoldByWeight`, `FoodProductSoldByVolume` en vous inspirant du schéma UML de la figure 1

   Le constructeur de la classe `ConsumerProduct` contiendra les paramètres pour le nom du produit et son prix hors taxes.

   (b) La taxe à valeur ajoutée de chaque produit de consommation est connue par la méthode `double getVAT()`. Celle-ci est par défaut fixée à 20% pour les produits de consommation courants, mais passe à 5,5% pour les produits alimentaires. **Redéfinir** la méthode `double getVAT()` dans `FoodProduct` pour provoquer cet effet.

   (c) Implémenter le constructeur de la classe `FoodProduct` pour ajouter la DLV comme **objet composé** de type `Deadline` aux produits alimentaires.

   (d) Implémenter le constructeur des classes `FoodProductSoldByWeight` et `FoodProductSoldByVolume` pour ajouter respectivement le poids et le volume du produit.

   (e) Nous serions tentés d'enregistrer à la fois le prix hors taxe au kilogramme des denrées alimentaires vendues au poids et le prix hors taxe d'une unité quand la marchandise est vendue par lots (yaourts, packs de lait, etc.)

   Expliquer pourquoi cette méthode est malheureuse.

   (f) **Redéfinir** la méthode `public double getUnitPrice()` dans `FoodProductSoldByWeight` pour prendre en compte dans le calcul le poids d'une unité de produit achetée.

   Idem dans `FoodProductSoldByVolume` pour prendre en compte dans le calcul le volume d'une unité de produit achetée.

   Explications complémentaires sur les méthodes :
   — double getPrice() ;
   Prix brut du produit hors tva (au kilo, au litre, à l'unité)

— double getVat() ;
Montant de la taxe à valeur ajoutée (5,5% ou 20%)
— double getVatPrice() ;
Prix TTC (avec la tva) du produit
— double getVatAmount() ;
Montant du coût de la taxe
— double getUnitPrice() ;
Prix HT d'un produit en vente unitaire
— double getUnitVatPrice() ;
Prix TTC (avec la tva) du produit en vente unitaire
— double getUnitVatAmount() ;
Montant du coût de la taxe du produit en vente unitaire
— Exemple : Lait en pack de 6 bouteilles de 50cl. Prix du lait HT : 1.45€/l

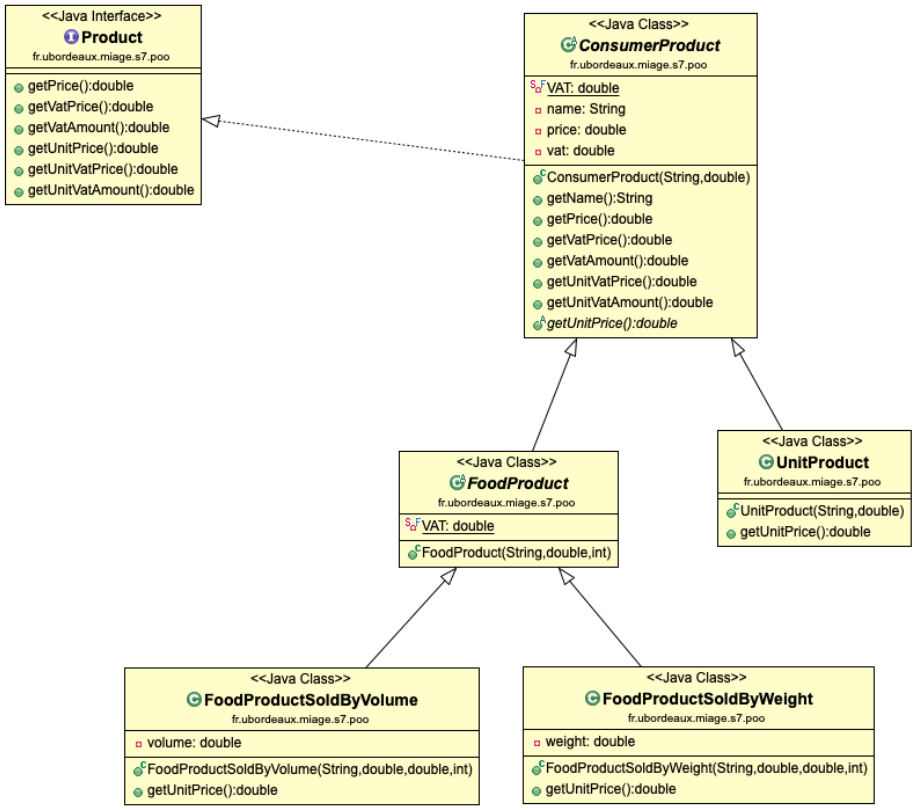

FIGURE 1 – Schéma UML des produits

2. **Interface**
Lors de la conception du projet, nous aimerions désigner le type abstrait de données pour des produits avec DLV. Quand un produit contient une telle date, nous aimerions pouvoir signaler le dépassement de la date limite de vente le cas échéant au moment de sa vente. Il

est important de ne jamais pouvoir modifier cette date et qu'elle soit fixée au moment où l'on stocke le produit en indiquant un nombre de jours entre maintenant et la DLV.

(a) Construire une interface `Dealine` qui contiendra la méthode
`boolean expired()`

(b) Implémenter la classe `DealineDays` qui fixe la date limite en jours à partir de l'instant où une instance est créée en donnant un nombre de jours qui restent avant expiration du produit.

(c) Modifier le code pour que `FoodProduct` implémente l'interface `Dealine` en **délégant** à un objet de type `Deadline` la gestion de la DLV.

La figure 2 est un schéma UML dont vous pouvez vous inspirer pour construire l'implémentation de la DLV.

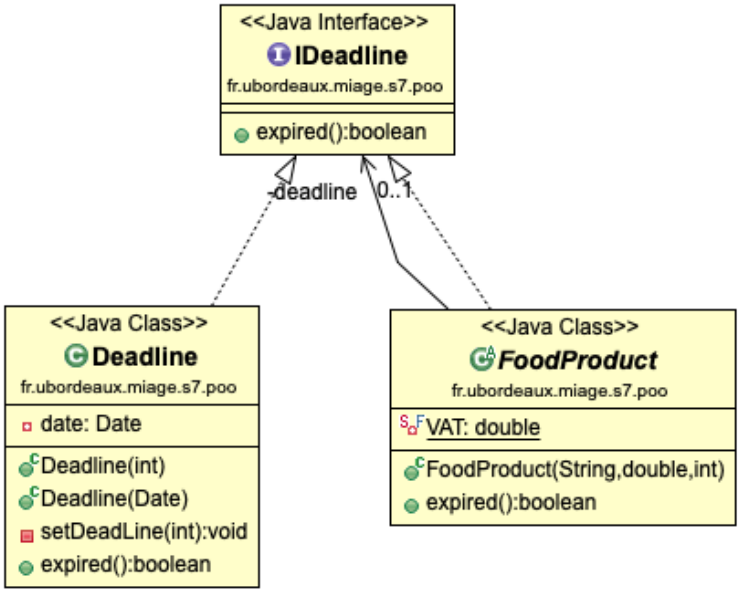

FIGURE 2 – Schéma UML de Deadline