TD/TP #1 & #2 - Patron de conception Stratégie & Singleton

Guillaume Santini

17 janvier 2024

1 Exercice principal : Production de factures

1.1 Modélisation UML

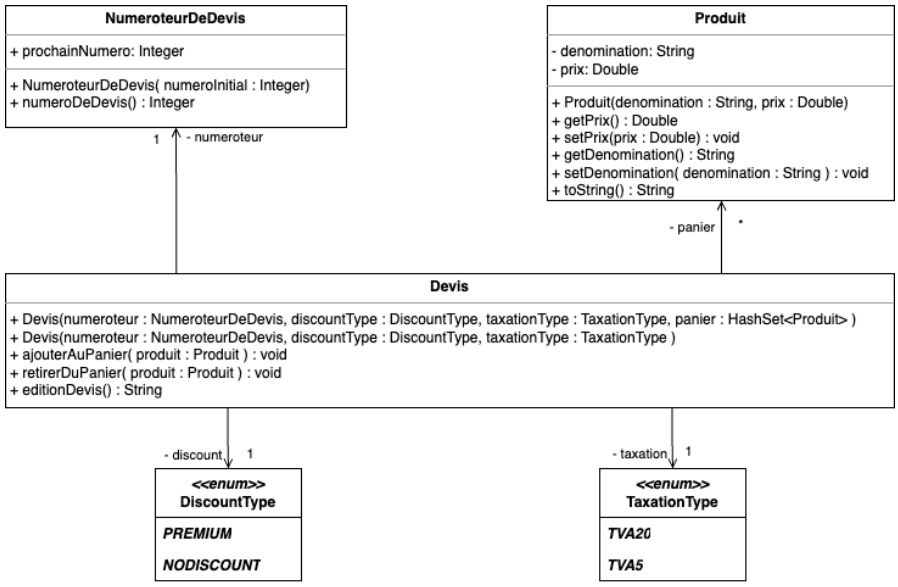
On souhaite développer un module d'édition de devis permettant de faire des simulations de factures en fonction du taux de TVA et de programmes de rabais. Une première phase de développement a donné lieu à la production d'une ébauche d'application constituée de trois classes (`Produit`, `NumeroteurDeDevis` et `Devis`) et deux énumérations (`TaxationType` et `DiscountType`).

Exercice 1 : Modélisation des entités de base

Proposez le diagramme de classe de la version initiale du module à partir de la documentation consultable ou des sources téléchargeables à partir de la page suivante :

— https://lipn.univ-paris13.fr/~santini/Patrons_conception/index.html

Correction :



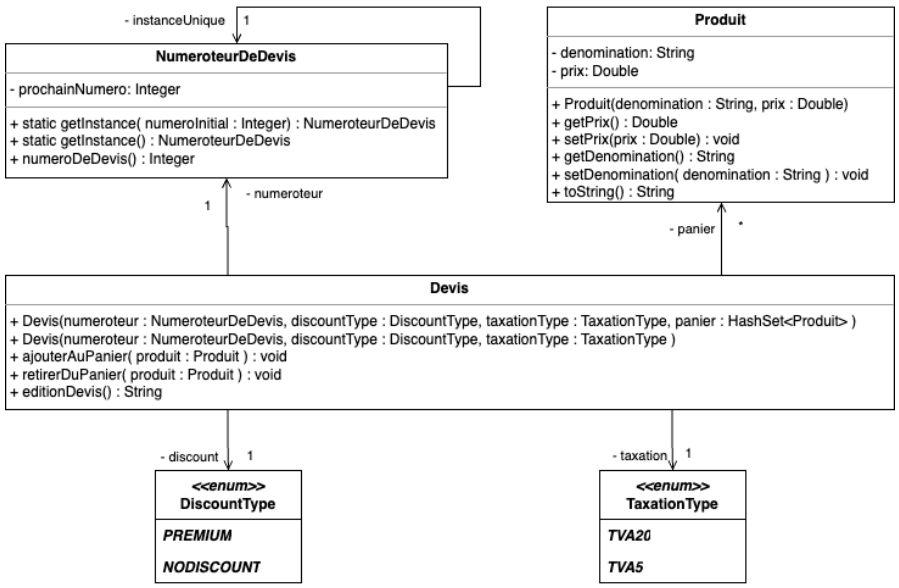
◇

Exercice 2 : Vers un modèle plus robuste

Afin de sécuriser l'application on veut garantir par le modèle que plusieurs objets de type `NumeroteurDeDevis` ne puissent être instanciés (ce qui pourrait rompre l'unicité des numéros de devis).

Modifiez le modèle pour garantir l'unicité de l'instance de `NumeroteurDeDevis`.

Correction :
Application du patron de conception Singleton sur la classe `NumeroteurDeDevis`.
Le polymorphisme sur la méthode `getInstance()` n'est pas exigée.



◇

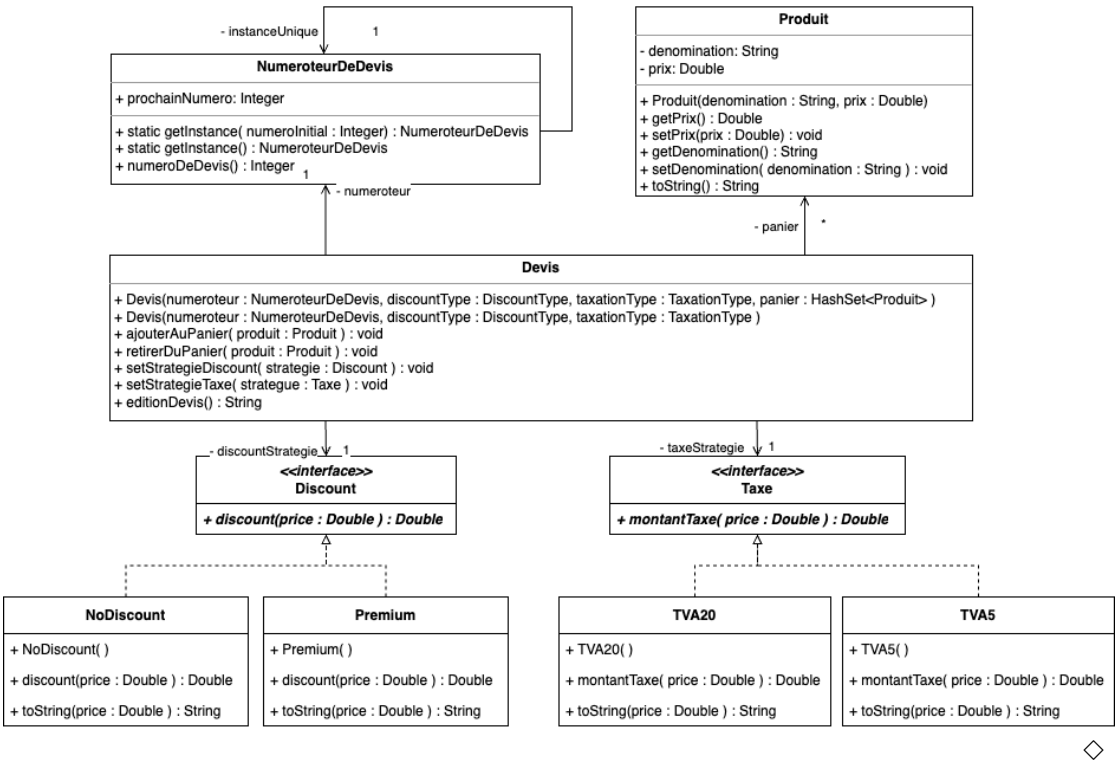
Exercice 3 : Intégration des principes SOLID

Ce modèle ne respecte pas les principes SOLID, notamment le principe Open/-Closed. La classe `Devis` est fortement couplée avec les énumérations `DiscountType` et `TaxationType`. L'ajout d'un nouveau mode de taxation (*e.g.* `DutyFree`) ou d'un nouveau programme de rabais (*e.g.* `PromotionTemporaire`) et leur prise en compte dans le calcul et l'édition du devis se ferait en effet au prix de la révision du code des deux énumérations (`DiscountType` et `TaxationType`) et du code de la méthode `editionDevis()` de la classe `Devis`.

Proposez une modification du modèle permettant :

— d'ajouter de nouveaux modes de taxation ou de nouveaux programmes de rabais,
— de pouvoir modifier à la volée des modes quand cela est souhaité.

Correction :



1.2 Implémentation Java

Sans qu'on ait à vous le préciser, à chaque TP vous produirez un code de qualité, c'est à dire documenté et dans la mesure du possible testé.

Exercice 4 : Récupération des classes déjà codées

Récupérez les sources des trois classes (`Produit`, `NumeroteurDeDevis` et `Devis`) et des deux énumérations (`TaxationType` et `DiscountType`)

— https://lipn.univ-paris13.fr/~santini/Patrons_conception/seance2/Facturation_ini/src.tgz

Faites un projet Éclipse (ou autre IDE de votre choix) et ajouter l'archive (Build Path→ add External Archives).

Exercice 5 : Implémentation du modèle

Traduisez la modélisation obtenue en code Java.

Attention : *Il ne faut pas tout coder et tester l'ensemble après. Il est impératif de tester chaque comportement un par un dans une ou plusieurs classes de test indépendantes.*

Exercice 6 : Exécution du modèle

Proposez un programme `Test.main()`principal qui respecte les spécifications suivantes :

1. La numérotation des devis commence à 200.
2. Le devis porte sur un panier contenant les produits suivant :
   — un "Vélo" à 400.€
   — des "Haltères" à 100.€
   — un Vetement - chaussures" à 40.€
   — un "Vetement - chaussettes" à 20.€
   — un "Complément alim." à 20.€

— un "Complément alim." à 20.€

3. Trois devis seront affichés :
   — un devis sans rabais avec une TVA à 20%,
   — un devis modifié avec avec le programme de rabais Premium,
   — un devis modifié avec une TVA à 5%.

Correction :

https://lipn.univ-paris13.fr/~santini/Patrons_conception/strategie/src.tgz

◇

Exercice 7 : Extension du modèle

Vérifiez que ce modèle vérifie bien le principe SOLID Open/Closed en rajoutant un programme de rabais temporaire (discount de 50%) et un mode de paiement en duty-free (0% de taxes).
Si votre modèle respecte bien le principe Open/Closed, ces modifications ne devraient pas vous amener à modifier les classes existantes...

2 Exercice optionel : Robots magasiniers

On souhaite modéliser un système robotisé de manipulation d'objets dans un magasin contenant des emplacements numérotés. Les robots doivent pouvoir attraper et déplacer les objets d'une emplacement à l'autre.

Le développement de l'application est contraint par les spécifications existantes du `Magasin` et des `Objetss`, dont des classes représentatives sont déjà développées.

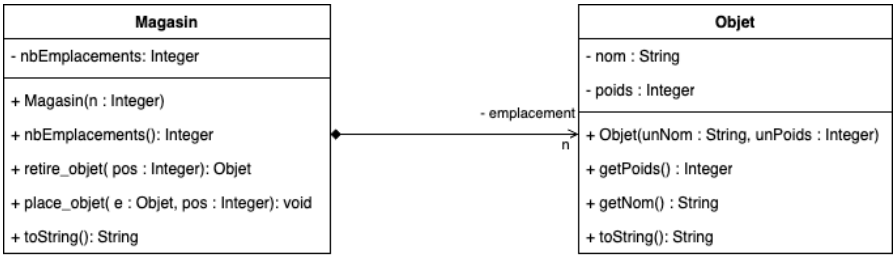
2.1 Modélisation UML

Exercice 8 : Modélisation des entités de base

Proposez le diagramme de classe de la version initiale du système à partir de la documentation consultable ou des sources téléchargeables à partir de la page suivante :

— https://lipn.univ-paris13.fr/~santini/Patrons_conception/index.html

Correction :



◇

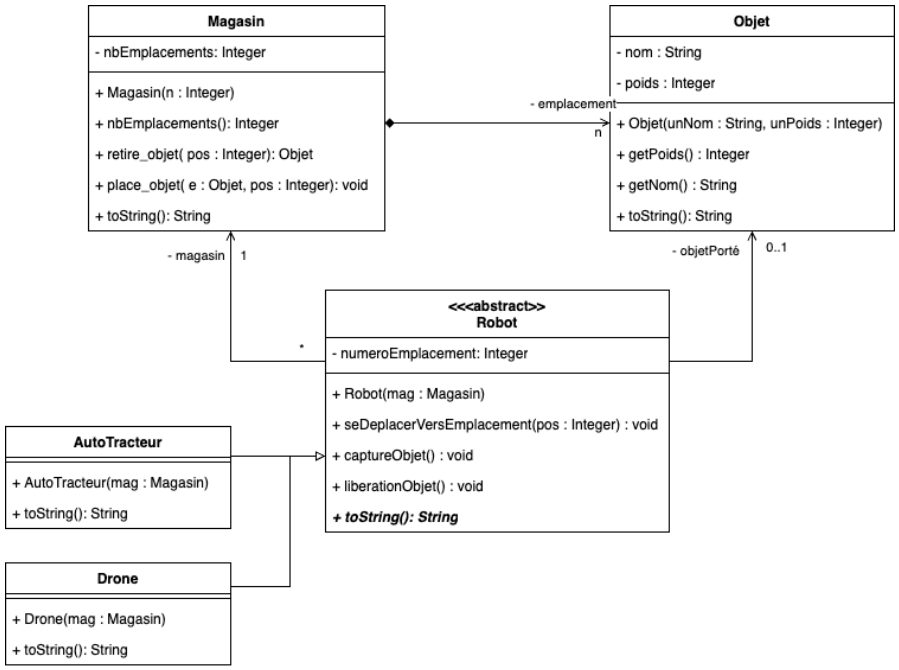
Exercice 9 : Modélisation du `Robot`

Complétez le modèle en décrivant la classe `Robot`. Pour cela on se donne les spécifications suivantes :

— Un `Robot` est toujours associé à un unique `Magasin`,
— Plusieurs `Robots` peuvent être associé à un `Magasin`,
— Un `Robot` est caractérisé par le numéro de l'emplacement devant lequel il se tient. À sa création, un `Robot` se situe devant la première case du `Magasin` et ne porte aucun `Objet`,

— Un `Robot` peut se déplacer d'un emplacement à l'autre dans le `Magasin`,
— Un `Robot` peut manipuler les `Objet`s du `Magasin` en les capturant ou en les libérant,
— Il existe deux types de `Robot`s, les `Drone`s et les `AutoTracteur`s.

Correction :



◇

Exercice 10 : Modélisation du `Robot`

Les `Robot`s sont dotés selon les modèles de différentes modes de déplacement et de différents modes de manipulation des `Objet`s.

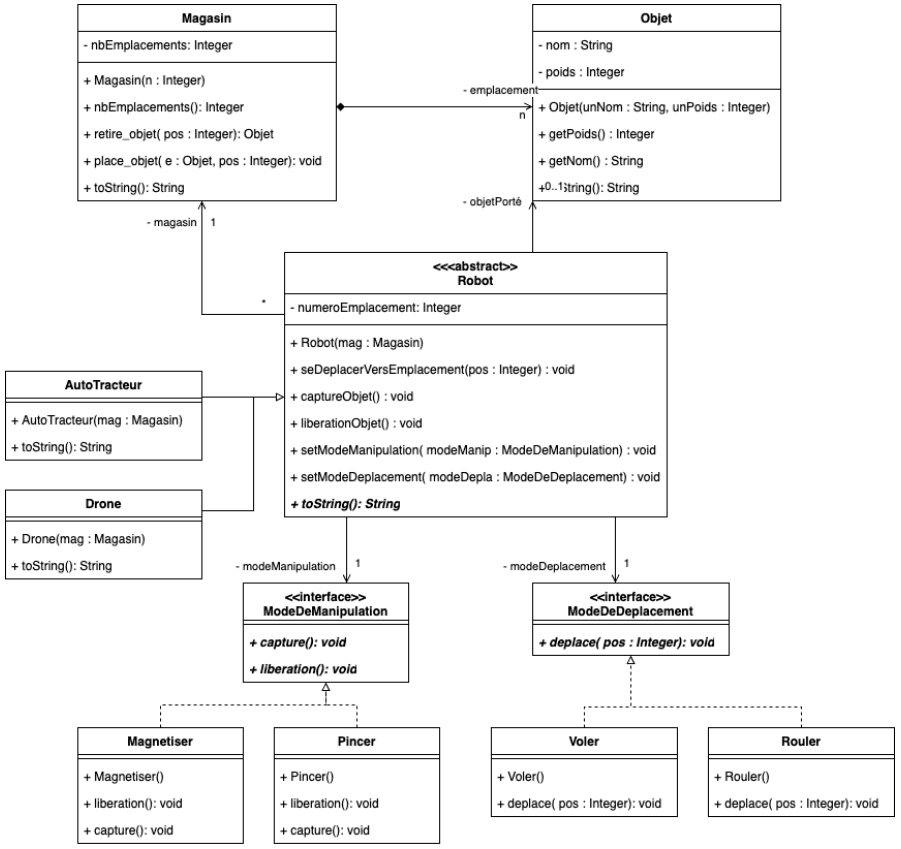
Les modes de déplacement d'un emplacement à l'autre sont :
— se déplacer en *roulant* ou
— se déplacer en *volant*.

Les modes de manipulation des `Objet`s sont :
— manipuler en *pinçant* les `Objet`s ou
— se déplacer en *magnétisant* les `Objet`s.

Complétez le diagramme pour tenir compte de ces paramètres. Ceux-ci doivent être ajustables en cours d'exécution.

Comme les spécifications d'implémentation de ces modes de manipulation et de déplacement ne sont pas encore connus, les actions de déplacement, de capture et de libération des `Objet`s ne se traduiront pour l'instant à l'exécution que par l'affichage de messages différenciant explicitement les actions effectuées.

Correction :



◇

2.2 Implémentation Java

Exercice 11 : Récupération des classes déjà codées

Récupérez les sources des deux classes (`Magasin` et `Objet`) :

— https://lipn.univ-paris13.fr/~santini/Patrons_conception/seance2/Robots_ini.tgz

Faites un projet Éclipse (ou autre IDE de votre choix) et ajouter l'archive (Build Path→ add External Archives).

Exercice 12 : Implémentation du modèle

Traduisez la modélisation finale obtenue en code Java.

Attention : *Il ne faut pas tout coder et tester l'ensemble après. Il est impératif de tester chaque comportement un par un dans une ou plusieurs classes de test indépendantes.*

Exercice 13 : Exécution du modèle

Proposez un programme `Test.main()`principal qui respecte les spécifications suivantes :

— création d'un `Magasin` de 3 *emplacements* contenant un unique `Objet` à la troisième case.

— création d'un `Drone`, se déplaçant en *volant* et manipulant les `Objets` par *magnetisme*,

— création d'un `AutoTracteur`, se déplaçant en *roulant* et manipulant les `Objets` par *pincement*,

— déplacement de l'`Objet` de la troisième case à la seconde (affichez l'état initial et final du casier) par le `Drone`,

— déplacement de l'`Objet` de la seconde case à la première (affichez l'état initial et final du casier) par l'`AutoTracteur`.
Vous ajouterez à la classe `Robot` les exceptions nécessaires pour gérer les situations conflictuelles.

Correction :

https://lipn.univ-paris13.fr/~santini/Patrons_conception/seance2/Robots.tgz ◇