Cours 420-266-LI Programmation orientée objet II Hiver 2024 Cégep Limoilou Département d'informatique Professeur : Martin Simoneau	TP3 (16 %) -Héritage – interface – collections -fichiers

Objectifs

- Réutiliser et rendre compatible en utilisant :
  - Héritage
  - Interface / Abstraction
  - Polymorphisme
- Fichiers
- Exceptions
- Collection
  - *List*
  - *Set*
  - *Map*

Contexte :

- Remettre votre projet complet sur Omnivox/*Léa* à la date indiquée.
- Le travail se fait en équipe avec le coéquipier qui vous a été assigné, la copie est interdite.

Contexte du projet

- Terminer le magasin commencé dans le TP2
  - Déterminer le coût de chaque produit en tenant compte des rabais accordés
  - Gérer l'achat des produits
  - Produire en continu un fichier d'historique des opérations de l'application.
  - Archiver les données dans des fichiers
  - Archiver et récupérer automatiquement les produits disponibles au début et à la fin de l'application
  - Produire le texte de l'onglet À propos à partir d'un fichier de ressource
- Vous devez avoir installé le JDK17 de Zulu pour pouvoir faire fonctionner ce TP : https://www.azul.com/downloads/?version=java-17-lts&os=windows&architecture=x86-64-bit&package=jdk-fx - zulu
- On vous a remis un projet avec cet énoncé. Le projet comprend plusieurs package:
  - Le package ***application*** contient tout le code qui lance l'application, gère l'UI et interagit avec votre code par l'intermédiaire de l'interface *Modele*. VOUS NE DEVEZ RIEN MODIFIER DANS CE PACKAGE!
  - Le package *client* contient plusieurs sous-package dans lesquels vous aller devoir travailler:
    - ***echange*** contient les interfaces qui permettent au UI d'interagir avec vos classes. Ne modifiez pas ces interfaces.
    - ***client*** contient les classes relatives au client du magasin: le panier, l'achat et la livraison. Il sera utilisé dans le TP3.
    - ***produit*** contient les classes relatives aux produits vendus par le magasin: les classes pour les boîtes et les classes pour les produits.
    - ***section*** contient les classes relatives aux différents endroits utilisés par le magasin: entrepôt, Vrac…
    - ***fichier*** contient les classes qui écriront et liront les fichiers nécessaires.
  - Pour lancer l'application vous devrez exécuter la méthode main qui se trouve dans la classe *tp2/application/MagasinApplication*.
  - Avec cet énoncé, vous avez reçu plusieurs fichiers. Placez-les aux endroits indiqués par le tableau suivant:

Fichiers reçus avec le TP	Où les placer
Dossier **application**	Tp3
Dossier **echange**	Tp3/
magasin.fxml	resources/tp3/application
APropos.txt	C'est à vous de trouver où le placer

Notez que les 2 parties sont relativement indépendantes et qu'elles peuvent être réalisées en même temps.

Partie 1 Prix d'achat

1. Pour cette partie vous aurez besoin de créer les classes
    a. client/**Achat**
    b. client/**Panier**
2. Le but de cette section est de créer la portion du programme où le client peut acheter des produits.
    a. La première étape à réaliser est de permettre à l'utilisateur de mettre des produits dans le panier du client. La méthode ***Magasin.mettreDansPanier*** réalise cette tâche. Mais avant de la programmer, vous devez créer votre classe *Panier* pour qu'il puisse contenir plusieurs produits. Noter que votre magasin n'aura qu'un seul panier à gérer (pour vous simplifier la tâche).
        i. Créer la classe ***Panier*** qui doit:
            1. Conserver les produits choisit par l'utilisateur en vue de l'achat.
    b. La méthode ***Magasin.retirerDuPanier*** permet de retirer des produits du panier. Attention, il faut veiller à remettre les produits dans la section où ils étaient. C'est à vous de trouver une stratégie pour retenir l'endroit où était le produit avant de le mettre dans le panier.
    c. La méthode ***Magasin.getContenuPanier*** doit retourner le contenu du panier pour l'affichage.
    d. La méthode la plus importante est certainement ***Magasin.acheterPanier*** qui permet à l'utilisateur d'acheter tout le contenu du panier. La méthode reçoit certaines informations importantes en paramètre (nom de l'acheteur, rabais global et date d'achat). Pour cela, il faut :
        i. Créer un objet *Achat*,
        ii. Déterminer le coût d'achat par l'objet Achat
        iii. Déterminer les rabais et produire le texte de facture pour retourner à l'interface
        iv. puis vider le panier.
    e. Créer la classe ***Achat*** qui doit:
            1. Conserver l'information importante de cet achat
                a. Le nom du client (acheteur)
                b. Le numéro de facturation (vous devez gérer ce numéro unique)
                c. Le moment de l'achat (*LocalDateTime*)

d. Les montants calculés
    i. Montant des taxes (14% pour le TP)
    ii. Montant des rabais
    iii. Montant brute
e. La liste des produits achetés

2. Calculer le coût du panier:
    a. La somme du coût de chaque produit dans le panier :
        i. Vous devez déterminer un coût pour chaque produit en fonction des attributs propre à ce dernier. Réutiliser le code au maximum. Vous pouvez modifier vos produits, mais vous ne pouvez pas changer la hiérarchie complètement.
    b. Moins le montant total des rabais. Il existe 2 types de rabais qui doivent être calculés:
        i. **Rabais global** : Le rabais global est appliqué sur le montant total de la facture. Il est déterminé en déplaçant le curseur correspondant sur l'interface
        ii. **Rabais de produit**. Appliquer un rabais sur *uniquement* 2 des produits disponibles qui n'ont pas un parent immédiat en commun (respectez *Liskov*). Le calcul de ce rabais se fait par le produit et ne dépend que du produit. Le rabais doit dépendre d'au moins un attribut du produit à rabais.
        iii. **Rabais de section**. Appliquer un rabais qui est calculé par la section dans un seul appel pour l'ensemble des produits de la section impliquée qui sont dans le panier.
            1. **Vrac :** On détermine d'abord un *pourcentage de rabais maximal de base*. On détermine ensuite la fraction du volume du vrac qui est achetée par le client. Pour calculer le rabais du vrac, il faudra donc multiplier le rabais maximal de base par la fraction du volume achetée puis par le prix total des produits achetés dans le vrac: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/time/LocalDate.html

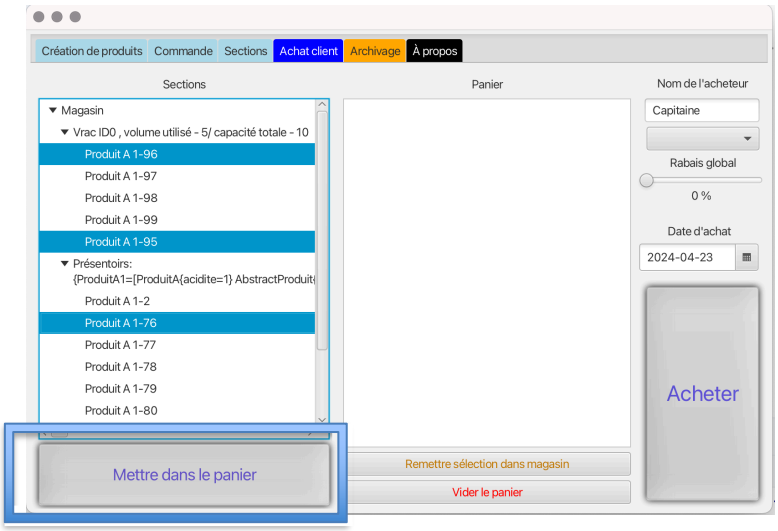
$$rabais\ vrac = \%rabais\_max \times fracion\_volume \times prix\_total$$

            2. **Presentoir :** Le rabais du présentoir dépend de la différence entre la date d'achat et la date du produit. Si un produit est acheté après un délai fixe (que vous devez déterminer ex : 4 jours) il obtient automatiquement un pourcentage de rabais fixe que vous devez également choisir. Le rabais du présentoir est la somme de tous les rabais des produits qui sont achetés après la date de rabais. Consulter l'API pour gérer la date
    c. Plus le montant des taxes appliquées sur tous les produits après rabais (14%)

d. N'oubliez pas de placer chacun des montants calculés dans les attributs correspondants de *Achat*.

3. Conserver une référence sur tous les produits qui ont été achetés.

f. Pour acheter l'utilisateur sélectionne les produits dans les différentes sections puis appuie sur le bouton ***Mettre dans le panier***



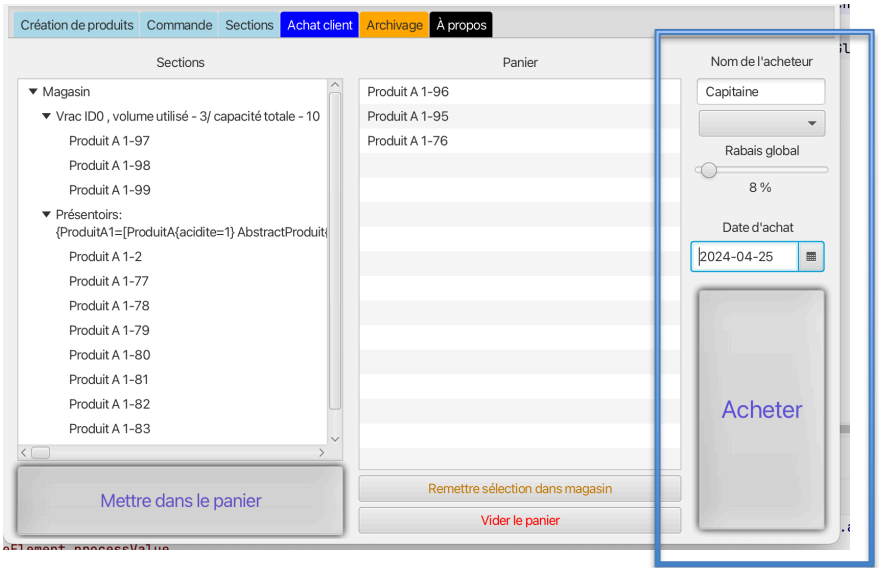
g. Pour acheter, l'utilisateur :

i. entre le nom de l'acheteur dans le champ à gauche (ou il sélectionne l'un des anciens acheteurs) ;

ii. sélectionne la date de l'achat;

iii. détermine le rabais global;

iv. appuie sur ***Acheter***.



h. L'achat est confirmé par un dialogue qui informe du prix et du nombre de boîtes utilisées. Votre méthode ***acheterPanier*** de Magasin doit retourner l'objet achat. La méthode *Descriptible.decrit* de ce dernier doit retourner une chaîne de caractères représentant la facture suivante :

i. Le nom de l'acheteur

ii. La liste des produits achetés avec leur prix

iii. Les montants

1. Le coût brut
2. Le rabais total accordé
3. Les taxes
4. Le coût total

iv. Le détail sur tous les rabais :

1. Rabais de produits avec prefix «***Rabais produit***»
2. Rabais de section avec prefix «***Section rabais***»
3. Rabais global.

N.B. pour formater les nombres avec 2 décimales, vous pouvez utiliser :

```
String.format("%.2f", valeur)
```

Partie 2 Persistence du magasin

IMPORTANT :

- Pour tout le code qui servira à écrire ou lire des fichiers, on vous demande d'essayer de réutiliser et de rendre compatible autant que possible.
- L'application doit elle-même créer tous les dossiers nécessaires lorsque ceux-ci sont absents.

1. Affichage de l'onglet ***À propos*** :

2. 

   a. Votre application doit avoir un fichier de ressources nommé ***APropos.txt*** qui est un template pour générer l'onglet *A Propos* de l'application. Si l'on modifie le texte dans le fichier *APropos.txt*, votre onglet *A Propos* devrait suivre la modification sans qu'on ait à toucher votre code.
      i. Placez le fichier de ressource au bon endroit dans votre application et gérez-le comme une ressource
      ii. Avec le TP3, la méthode ***init*** de l'interface *Modele* (implémenté par *Magasin*) a été modifié.
         1. En paramètre, vous allez recevoir un objet UI qui sera expliqué un peu plus loin dans ce document
         2. La méthode *init* doit également retourner une chaîne de caractères. C'est cette chaîne de caractères qui sera affichée dans l'onglet *A Propos.* La méthode init devra donc :
            a. *lire le fichier de ressource APropos.txt.*
            b. Ajouter au texte lu les noms des membres de l'équipe
            c. Retourner le tout.

3. Production de l'historique
   a. Créez une classe ***Historique*** qui doit ajouter en continu dans un fichier texte les événements qu'on lui transmet.
      i. L'historique doit être écrit dans le fichier ***historique.txt*** qui se trouve dans le dossier *historique* du dossier de travail de l'application. Idéalement, ouvrez et fermez le fichier à chaque nouvelle écriture. C'est plus simple à gérer
      ii. L'historique doit avoir une méthode pour ajouter des événements au fichier, un à la fois. Le format de l'événement doit être :
         1. **<date formaté> -> <texte de l'événement>**

            Exemple : ***25-04-24 : 07:17:30:91 -> Ouverture de l'application***

            Pour formater la date vous pouvez utiliser la méthode suivante sur un objet de type *LocalDateTime* :

```
unlocalDateTime.format(DateTimeFormatter.ofPattern("dd-MM-uu : hh:mm:ss:SS"))
```

      iii. L'historique devrait principalement être utilisé dans le magasin. L'application doit ajouter une entrée dans l'historique pour les événements suivants :

1. Ouverture de l'application
2. Création ou effacement d'un produit
3. Commande d'un ou plusieurs produits
4. Transfert de produits dans une section
5. Transfert de produit dans le panier
6. Achat d'un produit
7. Fermeture de l'application
8. Enregistrement du contenu du magasin

4. Archivage automatique des produits disponibles.
    a. On veut archiver automatiquement tous les produits créés par l'utilisateur (dans l'onglet Création de produit) dans un fichier nommé ***produits.mag*** qui doit être dans le dossier archive du répertoire de travail de l'application.
    b. Ce sont les méthodes ***init*** et ***stop*** de Magasin qui doivent respectivement lire et écrire les produits.
    c. Pour écrire et lire les fichiers, vous aurez besoin d'interagir avec l'UI pour connaître les différents produits que l'utilisateur a créés. À cette fin, la méthode init reçoit un objet de type UI qui vous permettra de connaître les produits disponibles (en appelant la méthode *getProduitsDisponibles*) ou de les spécifier (avec la méthode *setProduitsDisponibles*).
    d. Vous devrez donc, dans la méthode *stop,* mettre dans le fichier *produits.mag* tous les produits définis par l'utilisateur et, dans la méthode start, relire ce fichier pour les retransmettre au *UI*. Ainsi, l'utilisateur aura toujours ses produits disponibles et il n'aura pas à les recréer à chaque fois qu'il relance l'application.
    e. Notez qu'on a ajouté 2 boutons pour effacer les produits. Ils deviennent importants dans la mesure où l'application n'oublie plus les produits créés lorsqu'on la ferme. Les fonctionnalités d'effacement sont déjà entièrement gérées par le UI, vous n'avez rien à faire pour qu'elles fonctionnent.
5. Archivage des sections du magasin

1. Les trois boutons présentés sur l'image précédente permettent respectivement
    a. d'archiver le contenu du magasin,
    b. de remplacer le contenu du magasin parce qu'il a été mis dans le fichier d'archivage

c. De vider l'entrepôt et les sections client du magasin.

2. Les 3 boutons déclenchent respectivement les méthodes suivantes dans la classe magasin
   a. *archive()*
   b. *reconstruit()*
   c. *viderMagasin()*
3. On doit ajouter une fonctionnalité pour que l'application puisse sauvegarder le contenu du magasin sur le disque et pour le récupérer au besoin. Pour y arriver, vous devrez programmer les classes ***MagasinReader*** et ***MagasinWriter***. Ces classes doivent:
   a. Sérialiser et désérialiser le contenu du magasin. Vous pouvez choisir entre fichier texte, binaire ou objet. Comme la sérialisation objet est beaucoup plus simple, ceux qui effectueront la sérialisation binaire ou texte auront un extra de 5%. Notez également que les fichiers de sérialisation objet deviennent souvent incompatibles lorsque les classes impliquées dans l'enregistrement sont modifiées.
      i. L'entrepôt
      ii. Toutes les *AireI*
         1. *Vrac*
         2. *Aire de présentoirs*
   b. Il n'y a qu'un seul fichier d'archivage à la fois.

Répartition suggérée :

Semaines	Programmeur 1	Programmeur 2
Semaine 1	• Panier et Remise des produits : panier vers sections	• Classe Achat calcul des montants
	• Calcule du coup (ajustement hiérarchie) • Rabais produits	• Calcule des rabais (ajustement hiérarchie) • Rabais section
Semaine 2	• Historique	• Génération du ***A Propos*** dans magasin.init()
	• Lecture / écriture des sections et effacement du magasin	• Lecture / écriture des Produits disponibles

boubou boubou boubou boubou jerome jerome jerome jerome boubou

Pour programmeur sans équipe :

Semaines	Programmeur 1
Semaine 1	• *Panier* **sans** la remise des produits • Classe *Achat* calcul des montants
	• Calcule du coup (ajustement hiérarchie) • 2 Rabais produits (pas les rabais section)
Semaine 2	• Génération du ***A Propos*** dans *magasin.init()*

	• Lecture / écriture des Produits disponibles

Échéances

1. Semaine 1
    a. Panier et Achat
    b. Calcule des prix et des rabais
2. Semaine 2
    a. Écriture et lecture des produits et section
    b. Historique et génération du A propos
3. Semaine 3
    a. Débogage, compatibilité et réutilisation

Exigences:

Exigences

- Mettre vos noms en commentaires dans l'en-tête de chacune des classes dans lequel vous avez travaillé.
- Par restructuration, changer TP2 pour TP3. Exemple *: tp2-msd-psg* devient ***tp3-msd-psg***

  Critères d'évaluation :

Qualité du code

  - Commentaires pertinents et suffisants. Un développeur normal ne devrait pas mettre plus de 15 secondes pour comprendre ce que fait votre méthode.
  - Maximum 10 lignes de code par méthode;
  - Les noms des méthodes et des attributs sont pertinents et **complets**.
  - Il n'y a pas de code inutile ou commenté.
  - Formatage effectué dans chaque classe.
  - Algorithmes simples et efficaces. Toute méthode doit avoir au plus 10 lignes de code sans les commentaires.
  - Le code fonctionne sans erreurs.
  - Toutes les tâches demandées ont été accomplies.
- Classification et polymorphisme
  - La réutilisation et la compatibilité sont sans failles.
  - Les noms de classes, d'attributs et des méthodes sont pertinents.
  - L'annotation ***@Override*** est toujours utilisée lorsque c'est possible
  - L'abstraction est toujours utilisée lorsqu'elle est pertinente.

- Le code original est respecté (pas de modification inutile du code original, pas de changement de fonctionnalité) lorsque c'est demandé.

FIN