Cours 420-266-LI **Programmation orientée objet II** **Hiver 2025** **Cégep Limoilou** **Département d'informatique** **Professeur : Martin Simoneau** **Martin Couture**	**TP1 v-2.0 (10 %)** **Héritage - interface** **Polymorphisme** **Abstractions**

Objectifs

- Réutiliser et rendre compatible en utilisant :
  - Héritage
  - Interface
  - Délégation
  - Abstraction
  - Polymorphisme
  - Principes de programmation SOLID

Consignes importantes:

- Remettre vos projets sur Omnivox/*Léa* à la date indiquée.
- Le travail se fait seul.
- La copie est interdite. Tout code provenant d'une aide extérieure (ex.: ami, famille, *StackOverflow*, IA…) qui n'est pas mentionnée directement dans le code est considéré comme de la triche et occasionnera la note 0 pour le TP1. Vous n'aurez aucun point pour les parties faites par une aide extérieure.
- Vous devez pouvoir défendre **tout** le code qui se retrouve dans votre TP. Si l'enseignant doute que vous ayez réalisé le code vous-même, il pourra vous convoquer à une entrevue spéciale pour montrer que vous maîtrisez **tous les** éléments dans votre code. Un échec à cette entrevue entraînera la note 0 parce que votre code est alors considéré comme un plagiat.
- Les tuteurs du cégep n'ont pas le droit de vous assister pour faire les TPs.
- Toujours ouvrir le dossier qui contient le fichier pom.xml.
- **Surveillez le canal TP1 sur Teams pour des mises à jour ou des modifications de l'énoncé.**

Contexte :

Un restaurant a fait concevoir un système de commandes par un développeur junior. Ce dernier a abandonné le projet au moment ou il devait concevoir la classe *Commande* parce qu'il voyait son code devenir trop compliqué et parce qu'il sentait qu'il allait perdre le contrôle. Pour remédier à la situation, on a fait appel à un brillant étudiant de POOII pour réparer le travail et terminer le projet.
Le restaurant est réputé pour ses très nombreuses promotions et par leur variété !

À faire

- Garder une copie de sécurité du projet à titre de référence
- Vous pouvez modifier le code source, mais vous devez garder les fonctionnalités initiales intactes (sauf si des changements sont demandés). Dans le doute, faites valider votre idée par le professeur.
- Vous avez le droit de vous créer d'autres package au besoin.

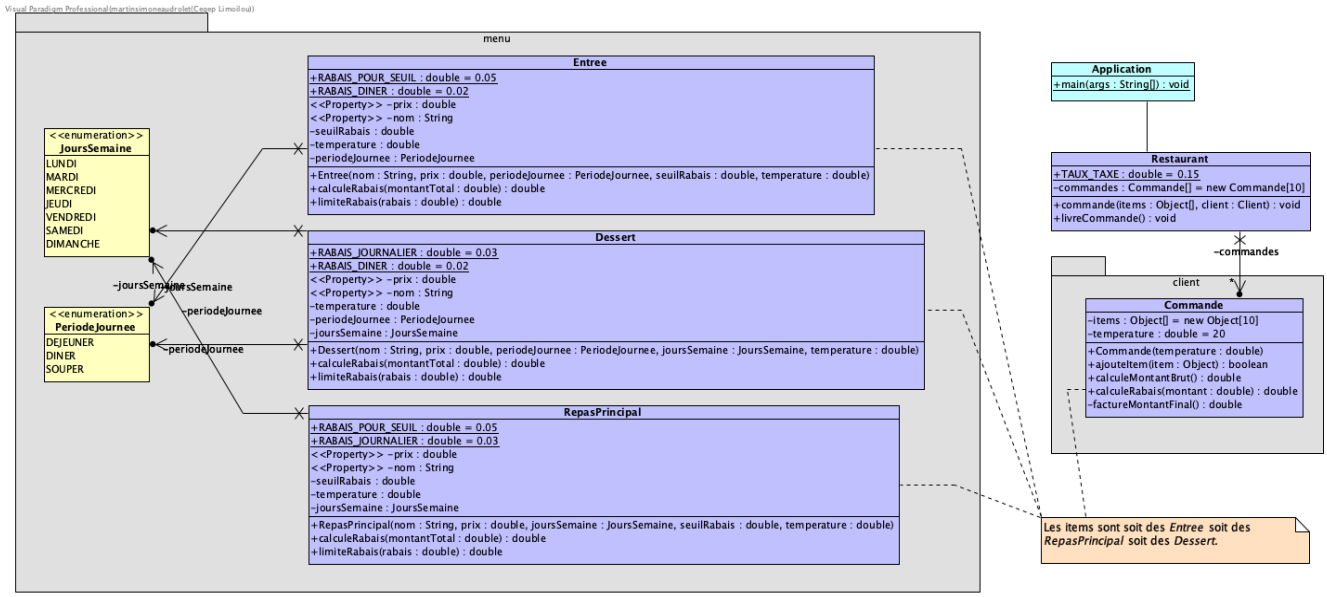

Figure 1Conception ratée à modifier

- Restructuration
    - Dans le package ***h25.msd.poo2***, vous trouverez la méthode *main* de la classe ***Application*** qui contient un usage typique des classes du restaurant. La classe principale est Restaurant est responsable de créer et de gérer les commandes. Une *commande* est un tableau contenant **entre 1 et 10 items**. Les items sont soit des instances de *Dessert*, *RepasPrincipal* ou *Entree*.
    - La première tâche consiste à améliorer le code :
        - Pour bien réutiliser le code qui se trouve dans les classes *Entree*, *Dessert* et *RepasPrincipale*. Attention à ne pas modifier les rabais qui doivent s'appliquer.

Type de rabais	**Entrée**	**Repas principal**	**Dessert**
Météo/température	Égale à la température lorsqu'elle est sous zéro. Rabais maximal de 30% (0 à 30%)		
Volume/seuil	Si le montant total de la facture dépasse un seuil prédéterminé, un rabais fixe de 5% s'applique		non
Horaire/période de la journée	2% au dîner	non	4% au souper
jour de semaine	non	Jeudi et vendredi 3% de rabais	
Client enregistré	(non lié au type d'item) il faut un nombre de 5 achats avant d'avoir le rabais. La hauteur du rabais peut être fixée en créant la commande.		

Figure 2 Tableau des rabais possibles en fonction du type d'item.

        - Pour rendre compatibles les méthodes *calculeRabais*() et *limiteRabais*().
        - Pour que l'appel de la méthode *calculeMontantBrut()* dans la classe commande soit beaucoup plus facile à faire et surtout que le code respecte le principe ***Open/closed***.
- Amélioration
    - Le développeur précédant n'a pas eu le temps de terminer les fonctionnalités. Il vous faudra ajouter les fonctionnalités suivantes :
        - Il existe un type de client qui est **enregistré**. Ce type de client obtient un rabais (qui peut être fixé à la construction de la commande) lorsqu'il a fait plus de 5 achats dans le restaurant. Vous devez modifier la commande pour gérer le client enregistré.
        - Il y a également 2 types de commandes qui impliquent des frais différents.

- Pour un repas en salle : le client peut laisser un pourboire (supérieur à 0%)
- Pour une commande à emporter : le client doit payer des frais fixes de livraison de 5$.

Frais	En salle	Pour emporter
pourboire	Associé à la commande	non
Frais de livraison	non	5 $

Figure 3 Frais en fonction du type de commande (livraison ou en salle)

- Attention de bien respecter les principes SOLID (voir critères d'évaluation).

- La classe Client :
  - La partie du simple client est déjà faite.
- La classe *Commande* est incomplète. Il faut :
  - Ajustez le ***constructeur*** au besoin.
  - ***ajouteItem(…)*** : la méthode n'est pas complète et elle ne fonctionne pas tout le temps. Vous pourrez changer sa signature au besoin.
  - ***calculeMontantBrute()*** : assurez-vous qu'elle fonctionne correctement
  - ***calculeRabais(double montantTotal)*** : Il faut la faire au complet. Elle doit sommer les rabais de chacun des items dans la commande avec le rabais de client enregistré et retourner le total.
- La classe *Restaurant* :
  - ***commande(…)*** : La classe doit effectuer une commande. Vous pouvez modifiez sa signature. Elle doit ajouter la commande dans le tableau des commandes si ce dernier n'est pas plein. Retourner faux si la commande n'a pas pu être saisie.
  - ***livreCommande()*** : Calcule le montant de chacune des commandes (montant brute, rabais et 5% de taxe). Elle doit également ajouter le montant final au client. Vous devez gérer le client vous-même. La méthode efface également les commandes du restaurant après les avoir traités.

- Validation :
  - Tests unitaires (à venir)

Critères d'évaluation et exigences:

Exigences

- Mettre vos noms en commentaires dans l'en-tête de chacune des classes dans lequel vous avez travaillé.
- Respectez la date de remise.

Qualité du code

- **Commentaires** pertinents et suffisants.
- Les **noms** des méthodes, des classes et des attributs sont pertinents et **complets**.
- Toutes les méthodes sont protégées par les **assertions** adéquates.
- Il n'y a **pas** de **code inutile** ou commenté.
- **Formatage** du code effectué dans chaque classe.
- **Algorithmes simples** et efficaces.

- Le code fonctionne **sans erreurs**.
- **Toutes** les tâches demandées ont été accomplies.
- Classification et polymorphisme
  - La réutilisation et la compatibilité sont sans failles. Respect des 3 premiers principes OO **SOL**ID (Single-responsibility, open-closed, Liskov substitution principle)
    - *Single responsibility*[1] : une classe doit avoir une seule raison d'être modifiée.
    - *Liskov Principle* : l'enfant doit assumer l'entièreté de ses parents (interfaces et classes)
    - *Open/Closed* : on ne devrait pas devoir modifier du code existant, mais il faut pouvoir ajouter de nouveau cas (nouveau type de commande (exemples : pour emporter, pour le bar express…) et de nouveau types de clients exemples : corporatifs, humanitaire…)
- L'annotation ***@Override*** est toujours utilisée lorsque c'est possible
- **L'abstraction/interface** est toujours utilisée lorsqu'elle est pertinente (abstract).
- Le **code original** est **respecté** (pas de modification inutile du code original, pas de changement de fonctionnalité), sauf lorsque c'est explicitement demandé.

Remise

- Remettre le projet complet archivé en ***zip*** et bien configuré sur Omnivox/LÉA. Une remise non conforme entraînera une pénalité. Une remise en retard entraînera 10% de pénalité par journée de retard conformément à la PIEA.
- Aucun autre moyen de remise n'est accepté.

FIN

[1] Robert C. Martin : " Gather together the things that change for the same reasons. Separate those things that change for different reasons."