Cégep André-Laurendeau

420-235-AL (Hiver 2023)

Évaluation Certificative

Orienté-Objet, Héritage, UML, Classe interne, Interface, Swing, Git, Entrées-Sorties

Échéance : 26 mai à minuit, ou tel qu'indiqué sur Léa

Michel Généreux; David Giasson

Objectif

Utiliser les concepts de programmation orientée-objet vus dans le cours pour compléter le logiciel fourni. Utiliser les diagrammes de classes UML correspondants pour guider votre travail. Accessoirement, utiliser la bibliothèque graphique Swing, les tests unitaires, lire et écrire dans un fichier de texte, réaliser une interface graphique, se servir du dépôt infonuagique GitHub et produire la documentation JavaDoc.

Directives importantes

- Ce travail pratique **doit être fait seul**.
- La remise se fera par Léa :
  - Zippez le répertoire qui contient le projet IntelliJ et l'historique git de votre code
  - Remettez ce fichier .zip sur Léa
  - [**Bonus**] Indiquez en commentaire sur la première ligne du fichier Main.java l'adresse URL de votre dépôt sur GitHub
- **La qualité du français est également importante**. Jusqu'à 10% de la note finale de votre travail pourrait être retirée pour cause d'un mauvais usage du français.
- Le travail doit être remis au plus tard à la date officielle indiquée sur Léa. Après cette date, 10% de pénalité sera enlevé par jour de retard jusqu'à un maximum de 5 jours de retard, après quoi la note 0 sera automatiquement attribuée.
- Bien entendu, toute forme de plagiat entrainera automatiquement la note zéro (0), ou pire encore. N'oubliez-pas d'indiquer vos références si vous utilisez des extraits de documents ou de code écrits par une autre personne.

Composition

Ce travail est constitué de **deux parties** distinctes qui doivent être réalisées **dans l'ordre.**

Critères d'évaluation

- Qualité du code (formatage, choix des noms de variables, etc.)
- Production des classes et méthodes conformément au diagramme UML
- Fonctionnement correct du programme
- Gestion appropriée des exceptions
- Interface graphique fonctionnelle
- Utilisation judicieuse de git et GitHub
- [**Nouveau**] Production de documentation JavaDoc (seulement pour les méthodes publiques, pas besoin de commenter les méthodes privées)

Résumé des actions requises pour compléter votre travail

1. Clonez le projet de départ depuis GitHub à l'adresse suivante : **https://github.com/alfclul/TS_H23.git**
2. Ouvrez et configurez le projet IntelliJ contenu dans le dossier cloné sur votre ordinateur.
3. À partir du projet IntelliJ de départ, complétez les seize (16) « *TODOs* » dans votre programme. L'ordre numérique place le code le plus creux (i.e. ne dépendant pas d'autres *TODO*) d'abord, ce qui peut faciliter le codage, mais vous pouvez bien sûr les compléter dans l'ordre que vous voulez. Des indices placés près des mentions *TODO* dans le code pourraient vous être utiles pour compléter vos tâches.
4. Tout au long de votre travail, effectuez régulièrement des *commit* dans votre repo git (localement seulement, pas de « push » vers GitHub exigé).
5. Remettez votre travail final sur Léa (un fichier zip contenant l'entièreté de votre projet).
6. [Bonus] Créez un dépôt privé sur GitHub pour votre projet, et copiez-y le code de votre travail. Inscrivez l'adresse (URL) de votre dépôt sur la première ligne de votre fichier source *Main.java* et invitez votre prof à ce repo pour obtenir quelques points bonus.

Préambule

Votre client gère une liste de « payables », qui peuvent être des personnes (employés) ou des « factures » d'achats à régler. Ces entités doivent faire l'objet d'un paiement à un moment donné (une **échéance** en jours). Le codage pour la gestion de ces payables représente la majeure partie de votre travail. Pour avoir une idée plus concrète des classes impliquées et de leurs interactions, veuillez vous référer au diagramme de classes fourni avec l'énoncé.

Les types de payables, accompagnés d'un exemple, sont présentés ci-dessous :

Facture

- Catégorie *Facture*
- ID *14*
- Numéro Pièce *34X53*
- Description *Tournevis*
- Nombre *34*
- Prix *23$*
- Mémo *Gros vendeur*
- Échéance *0*

Employé salarié

- Catégorie *EmployeSalarie*
- ID *10*
- Nom complet *Marie Renaud*
- Numéro d'AS *246864246*
- Salaire *5000$*
- Mémo *Bonne employée*
- Échéance *0*

Employé payé à l'heure

- Catégorie *EmployeHoraire*
- ID *11*
- Nom complet *Kevin Bouchard*
- Numéro d'AS *123321123*
- Taux horaire *25,50 $/hr*
- Heures travaillées *35 hrs*
- Mémo *Assidu*
- Échéance *0*

Employé salarié avec commission

- Catégorie *EmployeSalarieAvecCommission*
- ID *12*
- Prénom *Aline*
- Nom Propre *Brullemans*
- Numéro d'AS *123327832*
- Ventes *15000$*
- Taux de commission *0.1*
- Salaire de base *4000$*
- Mémo *Peu motivée*
- Échéance *0*

Employé horaire avec commission

- Catégorie *EmployeHoraireAvecCommission*
- ID *13*
- Prénom *Alan*
- Nom propre *Walsh*
- Numéro d'AS *973813265*
- Taux horaire *15 $/hr*
- Heures travaillées *32,50 hrs*
- Ventes *40000$*
- Taux de commission *0.15*
- Mémo *Du potentiel*
- Échéance *0*

Votre client voudrait un système avec les fonctionnalités suivantes :

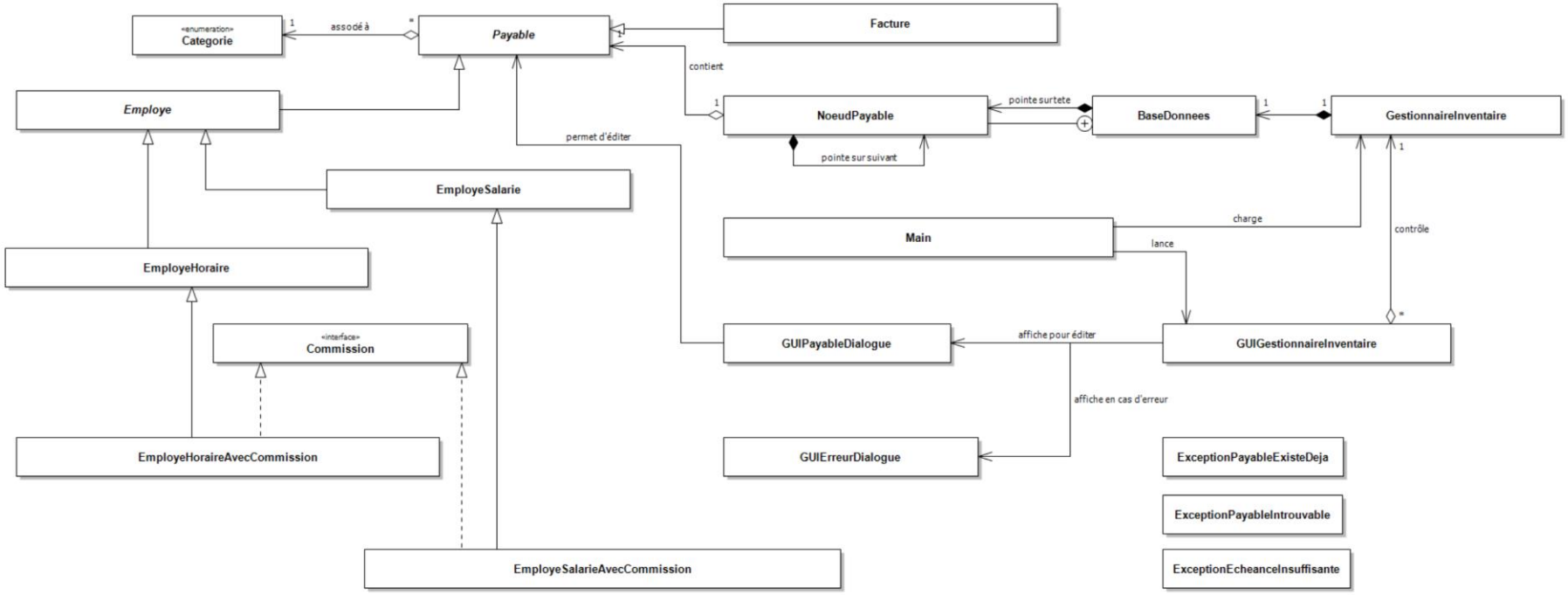
- Lire des payables à partir d'un fichier texte
- Sauvegarder des payables dans un fichier texte
- Ajouter un nouveau payable
- Supprimer un payable
- Incrémenter ou décrémenter l'échéance de paiement d'un payable
- Éditer les informations d'un payable
- Afficher une liste de tous les payables avec le montant dû

L'analyse des classes nécessaires a déjà été faite et une partie du code a déjà été écrite.

Partie 1 (80%)

Votre premier objectif est de programmer le code manquant (classes, méthodes et fonctionnalités) en vous basant sur les indications suivantes :

UML – le diagramme de classes du programme (version complète en annexe dans « **Diagramme de classes.pdf** ») :



À Faire – Assurez-vous de traiter tous les commentaires « **TODO** » se trouvant dans le code :

TODO	Fichier	Tâche	Contraintes et indices
1	EmployeSalarie	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	• Le salaire hebdomadaire doit être positif.
2	EmployeSalarie AvecCommission	Ajoutez tout le code nécessaire pour coder la classe au complet en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	• Le taux de commission doit être entre 0 et 1. • Les ventes brutes doivent être nulles ou positives. • Un tel employé est payé en ajoutant sa commission sur les ventes à son salaire hebdomadaire.
3	EmployeHoraire	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	• Les heures travaillées au-delà de 40 heures sont payées à 150% du taux horaire.
4	EmployeHoraire AvecCommission	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	• Le taux de commission doit être entre 0 et 1. • Les ventes brutes doivent être nulles ou positives. • Un tel employé est payé en ajoutant sa commission sur les ventes à son salaire horaire.
5	Facture	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	• Un prix par item (pièce) doit être positif. • Le total de la facture est donc fonction de la quantité et du prix par item.
6	BaseDonnees	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	Définir une classe interne **NoeudPayable** pour votre liste chaînée (simple) de payables.
7	GestionnaireInventaire	Ajoutez tout le code nécessaire en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	Il manque quelques méthodes pour gérer la base de données de l'inventaire. Attention aux exceptions!
8	Exception PayableIntrouvable	Ajoutez le code nécessaire pour créer l'exception générée quand on essaie de référer à un payable inexistant.	Inspirez-vous de l'exception déjà fournie!
9	Exception PayableExisteDeja	Ajoutez le code nécessaire pour créer l'exception générée quand on essaie de créer un payable avec un numéro de ID existant.	Inspirez-vous de l'exception déjà fournie!

TODO	Fichier	Tâche	Contraintes et indices
10	GUI GestionnaireInventaire	Ajoutez le code nécessaire pour augmenter d'un jour l'échéance de paiement ainsi que la gestion des erreurs possibles si nécessaire. L'information dans la liste des payables doit être automatiquement mise à jour.	1. Obtenir l'objet payable en inventaire à partir de l'ID du payable en surbrillance. 2. Augmenter son échéance. 3. Mettre à jour le *modeleListePayables*. Exception : payable introuvable
11	GUI GestionnaireInventaire	Ajoutez le code nécessaire pour réduire l'échéance paiement ainsi que la gestion des erreurs (il faut afficher un dialogue d'erreur si jamais on essaye d'aller en dessous de zéro). L'information dans la liste des payables doit être automatiquement mise à jour.	1. Obtenir l'objet payable en inventaire à partir de l'ID du payable en surbrillance. 2. Diminuer son échéance. 3. Mettre à jour le *modeleListePayables*. Exceptions : payable introuvable et échéance en-dessous de zéro
12	GUI GestionnaireInventaire	Ajoutez le code pour ouvrir le dialogue d'édition d'un payable ainsi que la gestion des erreurs possibles si nécessaire. L'information dans la liste des payables doit être automatiquement mise à jour.	1. Obtenir l'objet payable en inventaire à partir de l'ID du payable en surbrillance. 2. Afficher une boîte de dialogue **GUIPayableDialogue**. 3. Mettre à jour le *modeleListePayables*. Exception : payable introuvable
13	GUI GestionnaireInventaire	Ajoutez le code nécessaire pour supprimer un payable ainsi que la gestion des erreurs pour afficher un dialogue d'erreur si jamais on essaye d'effacer un payable sans faire de sélection dans la liste. L'information dans la liste des payables doit être automatiquement mise à jour.	1. Obtenir l'objet payable en inventaire à partir de l'ID du payable en surbrillance. 2. Le retirer de l'inventaire. 3. Mettre à jour le *modeleListePayables*. Exception : payable introuvable
14	Main	Codez la fonction `lireInventaire(String nomFichier, GestionnaireInventaire inventaire)`	Il faut créer un inventaire à partir de payables dans un fichier. Le format est celui de payables.in. Vous pouvez vous aider des regex qu'on retrouve dans les import fournies.
15	Main	Codez la fonction ecrireInventaire.	Il faut respecter le format de sauvegarde de payables.in ou payables.out.

Exceptions – Assurez-vous d'utiliser toutes les exceptions définies dans le diagramme UML aux endroits appropriés pour gérer les conditions d'erreur possibles (utilisez *throw / try / catch*).

Tests – Pour vous aider, le programmeur qui a fait l'analyse du système a mis en place un certain nombre de tests que vous trouverez dans la classe **Main**. Vous pourrez dé-commenter les lignes au fur et à mesure afin de tester ce que vous aurez programmé. Une copie des résultats attendus se trouve dans **tests-sortie-attendue.txt**.

Des tests unitaires (JUnit 4.13) vous sont également fournis pour chacun des *TODO* ainsi que pour chaque classe.

Partie 2 (20%)

Un autre programmeur a travaillé sur l'interface graphique. Maintenant que vous avez terminé le code de la partie 1, vous pouvez dé-commenter la dernière instruction du **Main** pour que votre programme lance l'interface graphique après les tests.

Ce court clip (mis à jour) montre le fonctionnement final attendu de l'interface.

Note – N'effacez pas les lignes des tests utilisés dans la partie 1, cela vous permettra de démarrer avec une interface qui contiendra déjà quelques items.

Le programmeur a presque tout fait, sauf les dialogues pour visualiser et éditer un item ainsi que les **TODO** 10, 11, 12 et 13 laissés dans **GUIGestionnaireInventaire.java**. C'est à vous que revient de finir cette tâche.

Barème de correction

Critères généraux	**20%**
Utilisation de git	5
Qualité du code	5
Respect du diagramme UML	10
Production de la javadoc	5

Partie 1	**60%**
TODO 01	5
TODO 02	5
TODO 03	5
TODO 04	5
TODO 05	5
TODO 06	5
TODO 07	5
TODO 08	5
TODO 09	5
TODO 14	5
TODO 15	5

Partie 2 – GUI	**20%**
TODO 10	5
TODO 11	5
TODO 12	5
TODO 13	5