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, lire et écrire dans un fichier de texte, réaliser une interface et se servir du dépôt infonuagique GitHub. Sans oublier l'utilisation de tests ainsi la documentation Javadoc.

Directives importantes

- Ce travail pratique **doit être fait seul**.
- La remise se fera par Léa ET GitHub:
  - Zippez le répertoire qui contient le projet IntelliJ et votre code
  - Remettez ce fichier .zip sur Léa
  - Fournir sur la première ligne du fichier Main.java l'adresse URL de votre dépôt GitHub
- *Note : pour tester votre fichier .zip (pensez au correcteur qui utilisera votre projet):*
  - Copiez le fichier .zip ailleurs sur votre ordinateur
  - Dé-zippez le fichier .zip
  - Ouvrez le projet dans IntelliJ et assurez-vous que tout fonctionne encore
- **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 à partir du diagramme UML
- Programmation des méthodes de chaque classe
- Fonctionnement correct du programme
- Gestion appropriée des exceptions
- Interface graphique fonctionnelle
- Utilisation judicieuse de GitHub

Résumé des actions requises pour compléter votre travail

1. Clonez le projet de départ sur GitHub à l'adresse suivante : https://github.com/alfclul/TS_H23.git
2. Créez un projet IntelliJ à partir du clone, c'est votre projet de départ.
3. Créez un dépôt GitHub pour votre projet. Inscrivez l'adresse URL de votre dépôt sur la première ligne de votre fichier source Main.java. Ce dépôt sera « cloné » par le correcteur lors de la correction. Idéalement, vous ferez un « push » pour chaque « TODO » complété dans votre travail.
4. 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.
5. Remettez votre travail sur LÉA (assurez-vous d'inclure un fichier « github.txt » contenant l'URL de votre repo en ligne si vous l'avez fait).

Préambule

Votre client gère une liste de « payables », i.e. des personnes (employés) ou des « factures » d'achat. Ces entités doivent faire l'objet d'un paiement (interface) 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 leur interaction, 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 motivé*
- 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 :

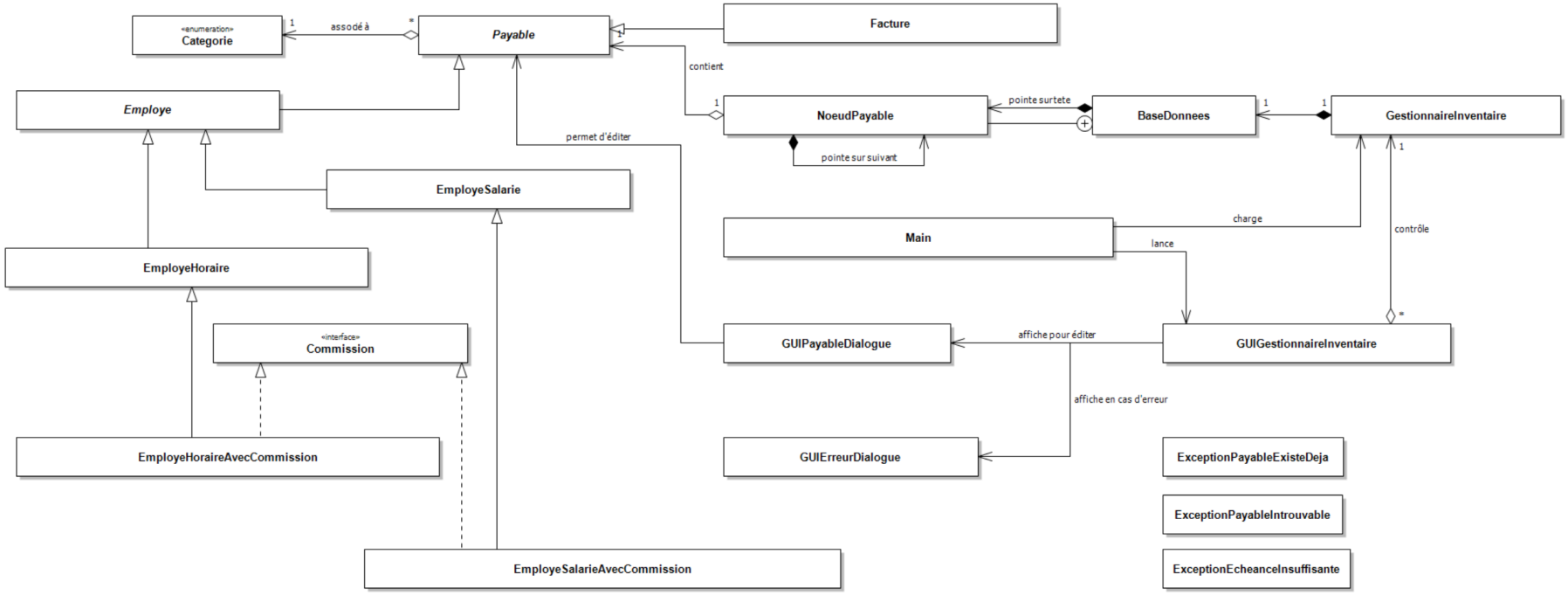
- Lire des payables à partir d'un fichier texte
- Sauvegarder des payables dans un fichier texte
- Ajouter un nouveau payable
- Supprimer un payable
- Incrémenter ou décrémenter l'échéance de paiement d'un payable
- Éditer les informations d'un payable
- Afficher une liste de tous les payables avec le 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	Tâche	Fichier	Contraintes et Indices
1	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	EmployeSalarieAvecCommission	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.
2	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	EmployeSalarie	Le salaire hebdomadaire doit être positif.
3	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.	EmployeHoraireAvecCommission	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.
4	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	EmployeHoraire	Les heures travaillées au-delà de 40 heures sont payées à 150% du taux horaire.
5	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	Facture	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	Ajoutez les méthodes nécessaires en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	BaseDonnees	Définir une classe interne NoeudPayable pour votre liste chaînée simples de payables.
7	Ajoutez tout le code nécessaire en vous basant sur le diagramme UML ainsi que la gestion des erreurs possibles si nécessaire.	GestionnaireInventaire	Il manque quelques méthodes pour gérer la base de données de l'inventaire. Attention aux exceptions!
8	Ajoutez le code nécessaire pour créer l'exception générée quand on essaie de référer à un payable inexistant.	ExceptionPayableIntrouvable	Inspirez-vous de la tâche 10 déjà fournie!
9	Ajoutez le code nécessaire pour créer l'exception générée quand on essaie de créer un payable avec un numéro de ID existant.	ExceptionPayableExisteDeja	Inspirez-vous de la tâche 10 déjà fournie!

10	Ajoutez le code nécessaire pour créer l'exception générée quand le nombre de jours d'échéance passe sous zéro.	ExceptionEcheanceInsuffisante	Celle-ci vous est fournie gracieusement.
11	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.	GUIGestionnaireInventaire	Obtenir le payable en inventaire à partir de l'ID du payable en surbrillance. Augmenter son échéance. Mettre à jour le modeleListePayables. Exception : payable introuvable
12	Ajoutez le code nécessaire pour réduire l'échéance paiement ainsi que la gestion des erreurs et afficher un dialogue d'erreur si jamais on essaye d'aller en dessous de zéro.	GUIGestionnaireInventaire	Obtenir le payable en inventaire à partir de l'ID du payable en surbrillance. Diminuer son échéance. Mettre à jour le modeleListePayables. Exceptions : payable introuvable et en-dessous de zéro.
13	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.	GUIGestionnaireInventaire	Obtenir le payable en inventaire à partir de l'ID du payable en surbrillance. Afficher une boîte de dialogue GUIPayableDialogue. Mettre à jour le modeleListePayables. Exception : payable introuvable
14	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.	GUIGestionnaireInventaire	Obtenir le payable en inventaire à partir de l'ID du payable en surbrillance. Le retirer de l'inventaire. Mettre à jour le modeleListePayables. Exception : payable introuvable
15	Codez la fonction lireInventaire.	Main	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.
16	Codez la fonction ecrireInventaire.	Main	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**.

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 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** 11, 12, 13 et 14 laissés dans **GUIGestionnaireInventaire.java**. C'est à vous que revient de finir cette tâche.

Barème de correction

Critères généraux		**20%**
Utilisation de GitHub	**5**	
Qualité du code	**5**	
Respect du diagramme UML	**10**	
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 10	**5 Fournie** 😊	
TODO 15	**5**	
TODO 16	**5**	
Partie 2 – GUI		**20%**
TODO 11	**5**	
TODO 12	**5**	
TODO 13	**5**	
TODO 14	**5**	

Travail de synthèse de substitution

Directives importantes

- Ce travail pratique **doit être fait seul**.
- La remise se fera par Léa ET GitHub:
  - Zippez le répertoire qui contient le projet IntelliJ et votre code
  - Remettez ce fichier .zip sur Léa dans le dossier TSsubs_H23
  - Fournir sur la première ligne du fichier Main.java l'adresse URL de votre dépôt GitHub **privé**. N'oubliez pas de m'inviter comme collaborateur (**alfclul**) sur ce dépôt.
- *Note : pour tester votre fichier .zip (pensez au correcteur qui utilisera votre projet):*
  - Copiez le fichier .zip ailleurs sur votre ordinateur
  - Dé-zippez le fichier .zip
  - Ouvrez le projet dans IntelliJ et assurez-vous que tout fonctionne encore
- **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.
- NOTE : ce travail est pour permettre à ceux d'entre vous (une majorité sans doute) qui n'auraient pas complété le travail original avant le mardi 16 mai 10h18 de le faire, ou même de le substituer à leur remise originale s'ils le souhaitent. Ceux qui sont satisfaits de leur travail original avant le 16 mai 10h18 n'ont donc, si c'est leur souhait, rien à remettre d'autre. En définitive, le seul interdit est de remettre des commit du travail original après le mardi 16 mai 10h18.

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 à partir du diagramme UML
- Programmation des méthodes demandées
- Fonctionnement correct du programme
- Gestion appropriée des exceptions
- Interface graphique fonctionnelle
- Utilisation judicieuse de GitHub

Résumé des actions requises pour compléter votre travail

1. Clonez les fichiers de départ sur GitHub à l'adresse suivante : https://github.com/alfclul/TSubs_H23.git
2. Créez un projet IntelliJ à partir des fichiers sources, c'est votre projet de départ.
3. Créez un dépôt GitHub **privée** pour votre projet et invitez-moi comme collaborateur (**alfclul**). Inscrivez l'adresse URL de votre dépôt sur la première ligne de votre fichier source Main.java. Ce dépôt sera « cloné » par le correcteur lors de la correction. Idéalement, vous ferez un « push » pour chaque « TODO » complété dans votre travail.
4. À partir du projet IntelliJ de départ, complétez les seize TODOs manquant 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. Au final, vous devez compléter les seize (16) TODOs, soient à partir du travail original seul, soit à partir du travail de substitution seul ou soit à partir d'un mélange des deux.
5. Remettez votre travail sur LÉA ainsi que la « **Nouvelle grille de correction** » complétée pour nous permettre de corriger les TODOS de votre choix.

Préambule

Ce travail de substitution est en fait une série de TODOs ciblés sans que le tout constitue un travail homogène avec une tâche circonscrite bien précise. L'idée ici est d'évaluer les mêmes notions visées par le travail original par des questions ciblés.

Partie 1 (80%)

À l'aide des diagrammes de classe et du code de départ, votre tâche est de compléter le code départ fournie. Les **TODO** 1 à 10, 15 et 16 se réfèrent à cette tâche.

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 décommenter les lignes au fur et à mesure afin de tester ce que vous aurez programmé. Les résultats attendus se trouvent commentée dans **le Main**.

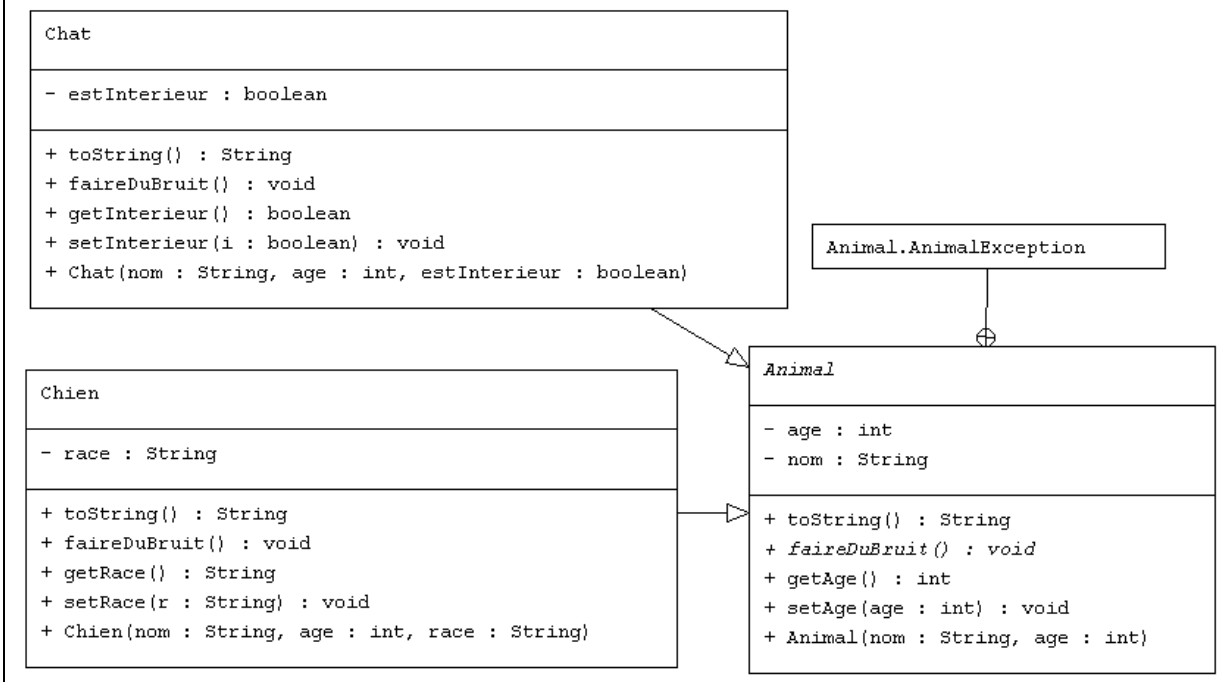
Partie 2 (20%)

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

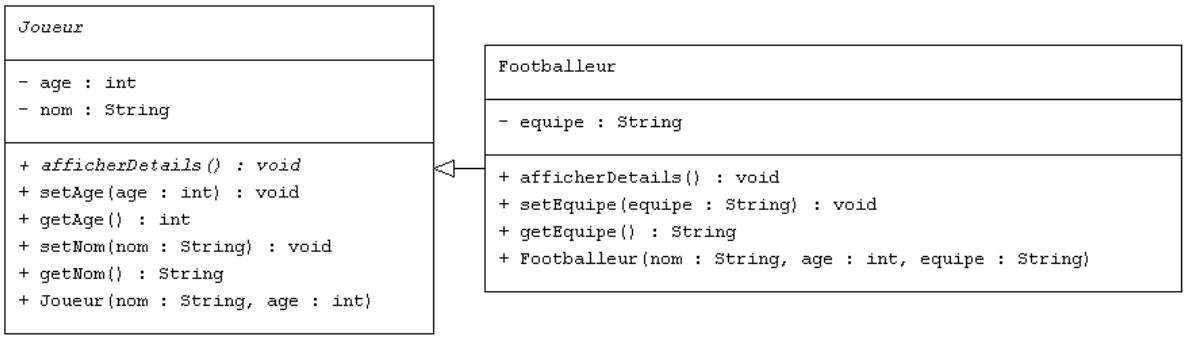
Les **TODO** 11, 12, 13 et 14 se réfèrent à cette tâche.

<u>TODO de substitution</u> : à partir du fichier de départ et des diagrammes suivants, complétez les TODOs pour que la méthode main() fonctionne correctement.

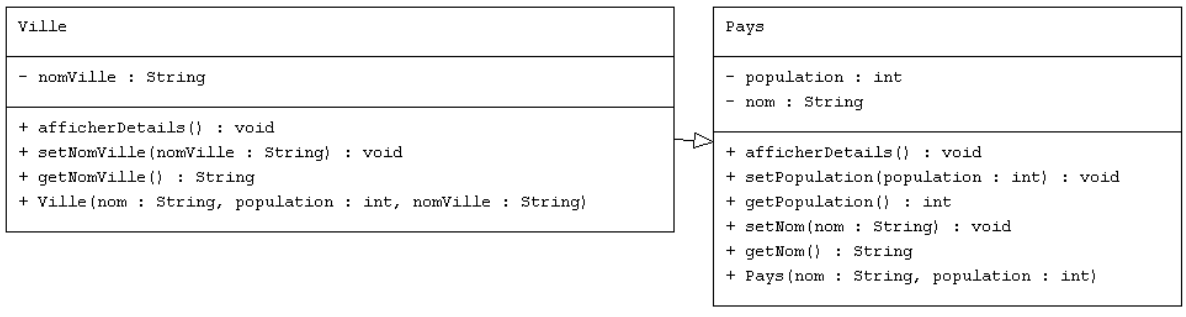
TODO	CLASSES
1.1 et 1.2	**Chien, Chat, Animal**



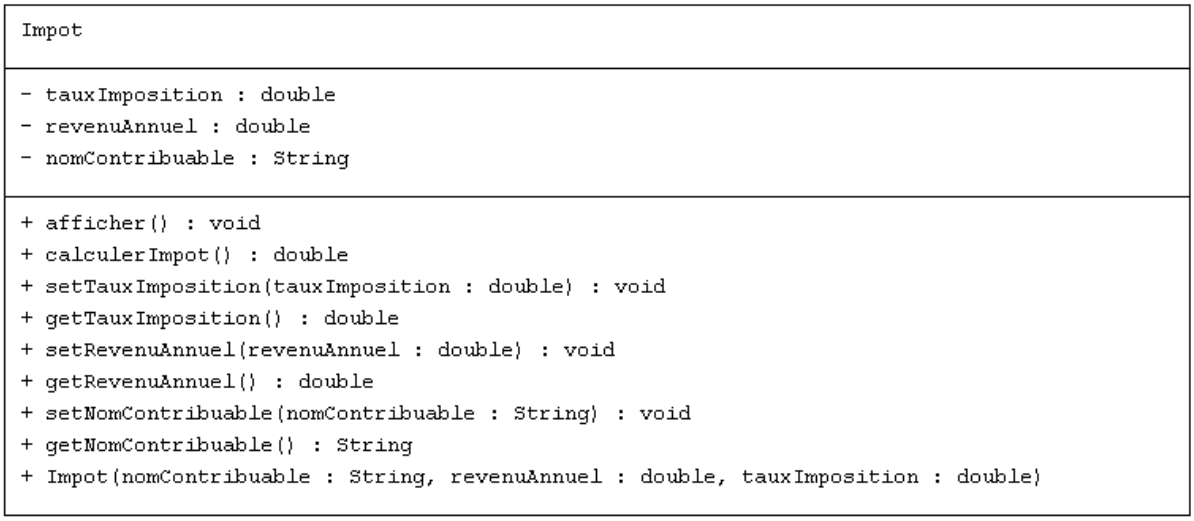
2	**Footballeur, Joueur**



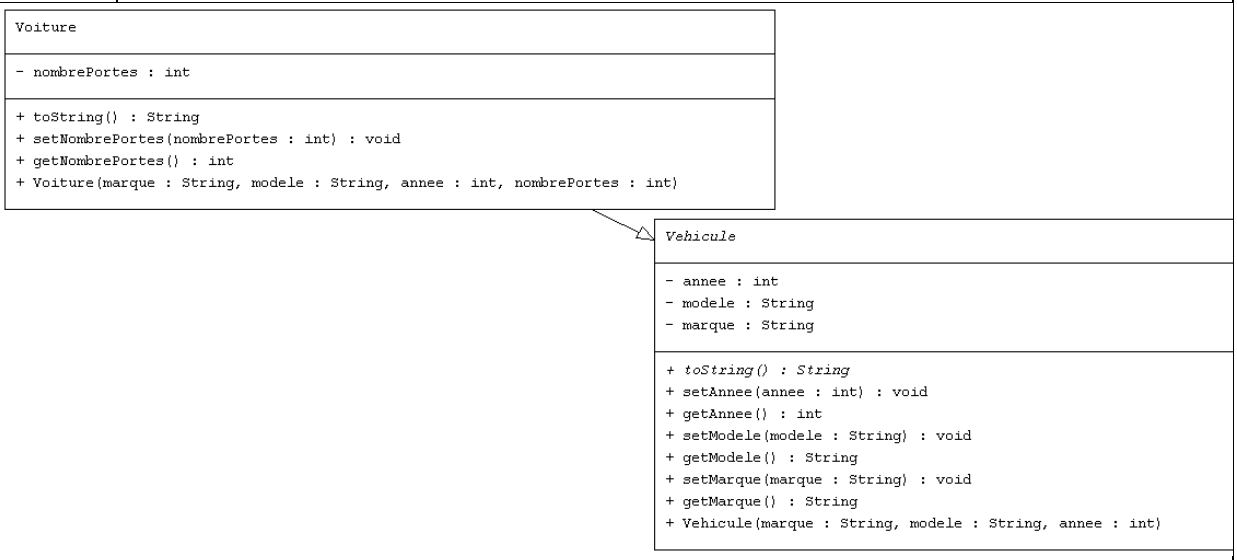
3	**Pays, Ville**



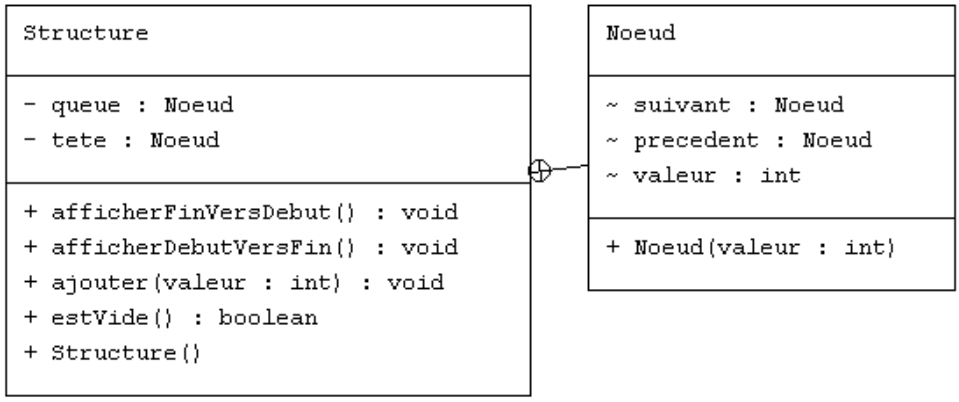
4 Impot



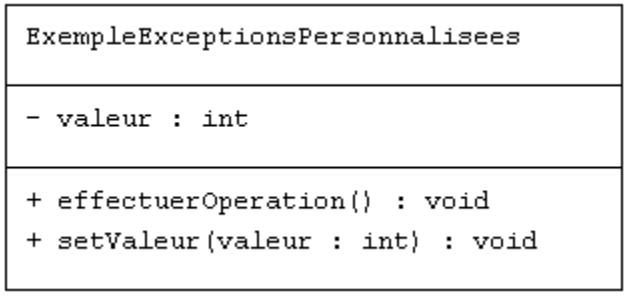
5 Vehicule, Voiture



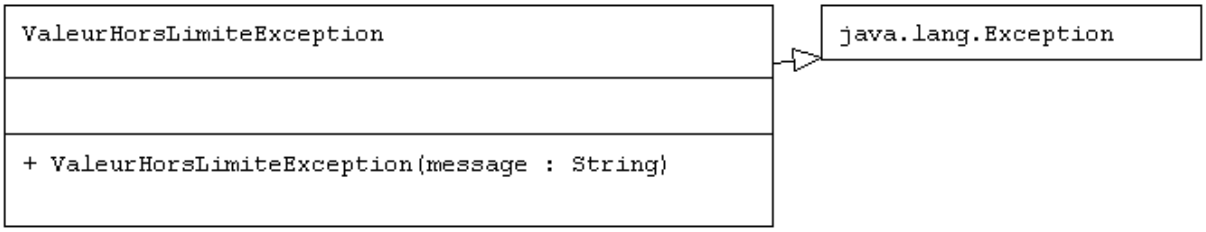
6 Structure, Noeud



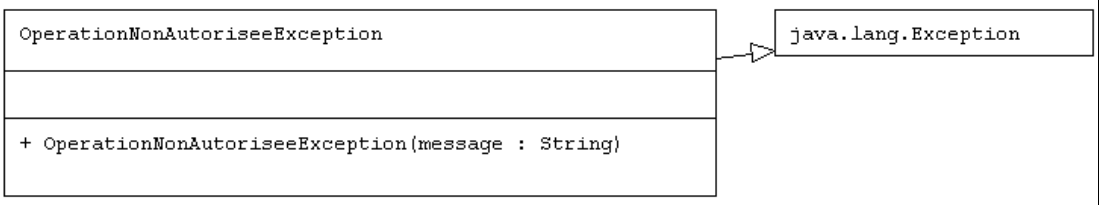
7	**ExempleExceptionsPersonnalisees**



8	**ValeurHorsLimiteException**



9	**OperationNonAutoriseeException**



10	**Rien à faire**
11	À venir
12	À venir
13	À venir
14	À venir
15	À venir
16	À venir

Nouvelle grille de correction **à compléter**	**Travail original** *Tous les TODO commis avant le 16 mai 10h18*	**Travail de substitution** *Tous les TODO commis après le 16 mai 10h18*	
Critères généraux			**25**
Utilisation de GitHub	Fournir votre adresse Github privée. N'oubliez pas de m'inviter comme collaborateur (alfclul)	Fournir votre adresse Github privée. N'oubliez pas de m'inviter comme collaborateur (alfclul)	**5**
Qualité du code	**TODOs sélectionnés plus bas.**		**5**
Javadoc	Seulement les méthodes publiques avec @param et @return plus une fois @author et @version. Indiquez quelle(s) version(s) à corriger.		**5**
Respect du/des diagramme(s) UML	**TODOs sélectionnés plus bas.**		**10**
Partie 1			**55**
TODO 01	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 02	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 03	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 04	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 05	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 06	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 07	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 08	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 09	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 10			**Fournie**
TODO 15	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 16	**Celui-ci?**	**Ou celui-là?**	**5**
Partie 2			**20**
TODO 11	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 12	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 13	**Celui-ci?**	**Ou celui-là?**	**5**
TODO 14	**Celui-ci?**	**Ou celui-là?**	**5**

Exemple : *vous avez eu le temps de compléter 5 TODOS sur le travail original et de le commenter avant le mardi 16 mai 10h18 et le reste vous l'avez complété sur le travail de substitution. Vous nous remettez donc la grille suivante sur LÉA avec votre travail:*

	Travail original *Tous les TODO commis avant le mardi 16 mai 10h18*	**Travail de substitution** *Tous les TODO commis après le mardi 16 mai 10h18*	
Critères généraux			**25**
Utilisation de GitHub	https://github.com/vous/TSori_H23 (ceci est un exemple)	https://github.com/vous/TSubs_H23 (ceci est un exemple)	**5**
Qualité du code	**TODOs sélectionnés plus bas.**		**5**
Javadoc	**√**		**5**
Respect du/des diagramme(s) UML	**TODOs sélectionnés plus bas.**		**10**
Partie 1			**55**
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 10			**Fournie**
TODO 15		**√**	**5**
TODO 16		**√**	**5**
Partie 2			**20**
TODO 11		**√**	**5**
TODO 12		**√**	**5**
TODO 13		**√**	**5**
TODO 14		**√**	**5**