Location de véhicules

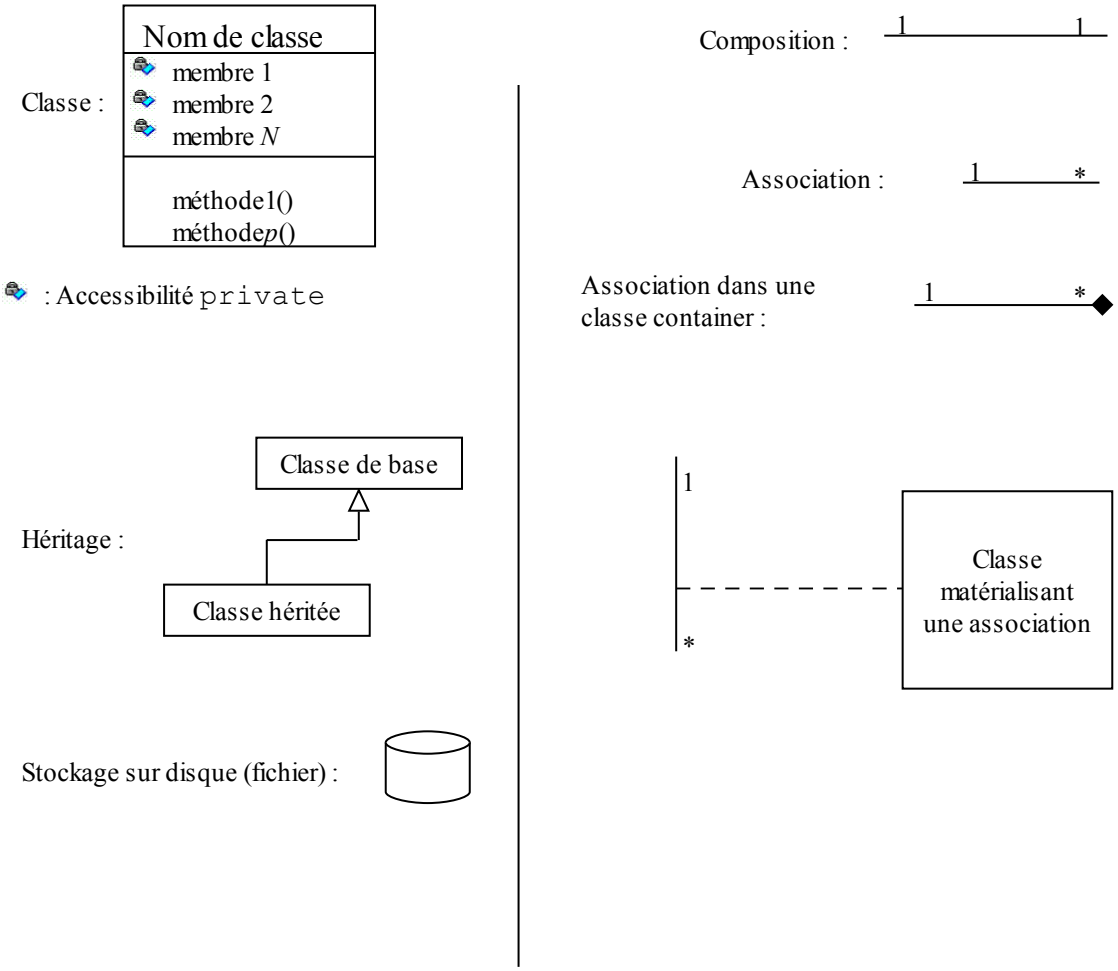
APPLICATION LOCATION DE VEHICULES

I Introduction

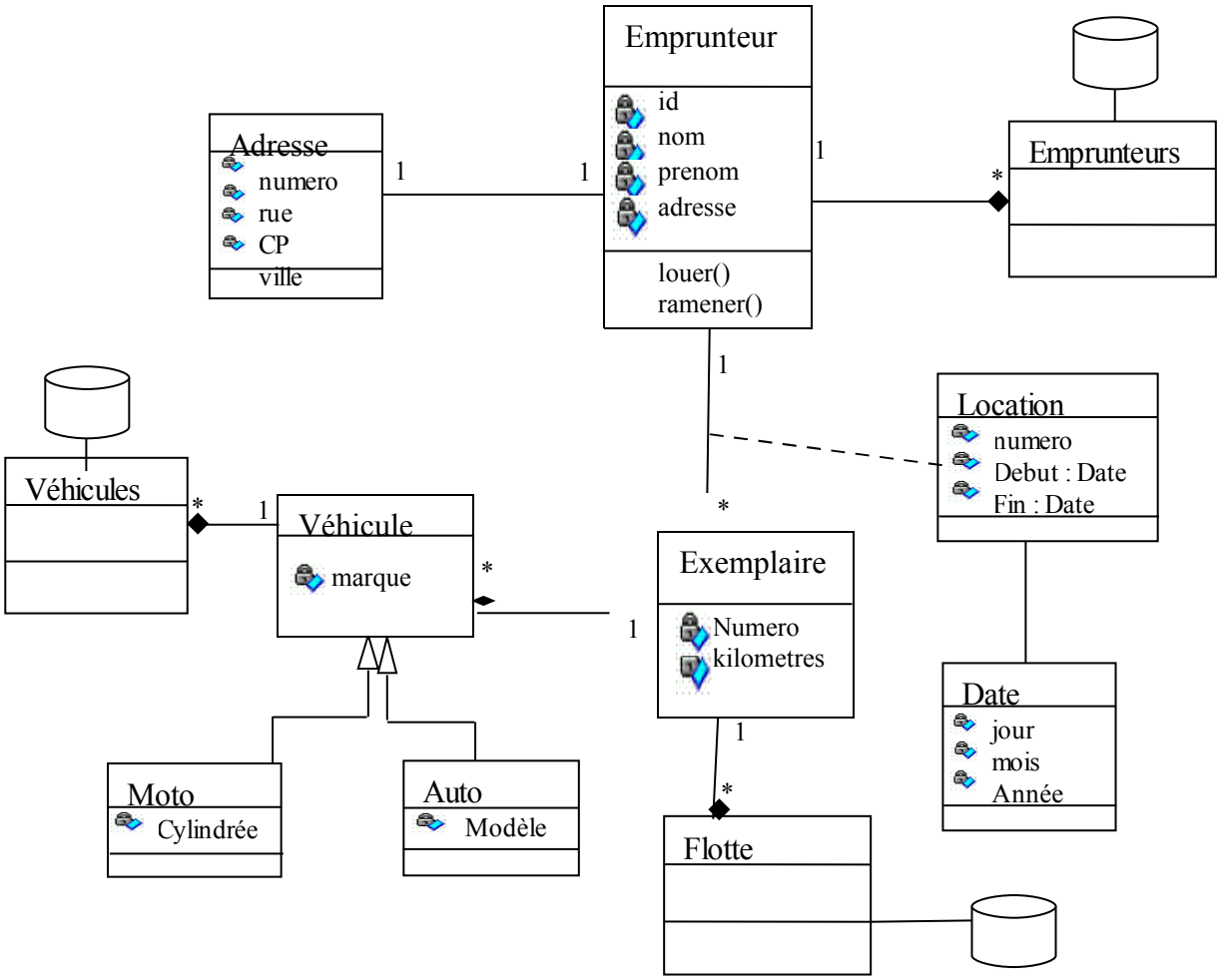
Le but de ce projet est de réaliser une application Orientée Objet pour la gestion d'une flotte de véhicules de location

Pour cette application, on vous fournit un diagramme des classes résultant d'une analyse préalable des besoins du système d'informations du loueur à informatiser.

La nomenclature utilisée pour la rédaction de ce diagramme est la suivante :



II Diagramme des classes utilisé pour l'application LOCATION



Bien évidemment, la liste des membres et méthodes apparaissant sur le diagramme des classes est loin d'être exhaustive : il vous appartiendra de l'enrichir et d'en fournir une version complète avec le rapport de projet.

III Cahier des charges de l'application LOCATION

1) Gestion des devis/locations

a) Toute location est matérialisée sous la forme d'un devis comportant les informations suivantes :

Informations complètes sur l'emprunteur, sur le véhicule, durée prévue du prêt.

En fonction de la durée du prêt et de la catégorie du véhicule, LOCATION produira un devis. Il vous appartient de définir les différents tarifs appliqués.

On supposera dans un premier temps que tous les véhicules d'un contrat de location ont la même date d'emprûnt et de retour.

b) LOCATION propose en option une assurance contre la dégradation et les accidents, à un tarif que vous choisirez.

c) Tout véhicule loué est loué pour une certaine durée avec un tarif convenu à l'avance. Au retour du véhicule, LOCATION doit éditer une facture tenant compte des points suivants :

- la durée effective de location: si le véhicule est rendu en retard par rapport à la date convenue, une pénalité (que vous définirez) sera appliquée;
- la consommation de carburant : un véhicule est prêté avec le plein de carburant. Selon le niveau de carburant restant (vide, ¼ du réservoir, ½ du réservoir, ¾ du réservoir), un supplément sera appliqué;
- l'état du véhicule : si aucune assurance n'a été souscrite par l'emprunteur et que le véhicule est endommagé, l'emprunteur aura à payer des frais de remise en état du véhicule, indiqués sur la facture. Il vous appartient de définir la manière dont seront stockées les informations sur l'état du véhicule.

2) Gestion des Ressources

a) LOCATION doit proposer tous les types de recherche suivants pour les ressources :

- Recherche par marque
- Recherche par kilométrage

b) LOCATION doit pouvoir insérer, supprimer ou modifier une ressource.

c) LOCATION doit proposer tous les types d'affichage suivants pour les ressources

- Liste par type : moto, voiture standard, voiture de luxe

d) LOCATION doit pouvoir trier les ressources selon leur marque et selon leur kilométrage.

3) Gestion des emprunteurs

a) LOCATION doit proposer tous les types de recherche suivants pour les adhérents (emprunteurs) :

- Recherche par nom
- Recherche de tous les emprunteurs empruntant un véhicule particulier.

b) LOCATION doit pouvoir insérer, supprimer ou modifier un adhérent (emprunteur).

c) LOCATION doit proposer un affichage de la liste des adhérents.

d) LOCATION doit pouvoir trier les adhérents selon leur nom et selon le montant total de leurs factures de tous les véhicules loués.

4) Gestion des locations

a) LOCATION doit proposer tous les types de recherche suivants pour les locations :

- Recherche par date de début
- Recherche de tous les emprunteurs ayant une location en cours

b) LOCATION doit pouvoir insérer, supprimer ou modifier une location.

c) LOCATION doit pouvoir afficher un devis et une facture.

d) LOCATION doit proposer un affichage de la liste des locations.

d) LOCATION doit pouvoir sauvegarder dans un fichier texte et un fichier pdf les factures des adhérents (un fichier par facture!).

Pour la génération des pdf, vous pourrez consulter la ressource suivante :

http://www.vogella.com/tutorials/JavaPDF/article.html

e) LOCATION doit pouvoir trier les adhérents selon le montant total de toutes leurs factures.

Indication pour les tris:

Les interfaces de comparaison **Comparable<T>** et **Comparator<T>** sont à utiliser.

ainsi que, si nécessaire, la méthode **Collections.sort.**

Il faudra bien réfléchir à la classe de l'interface Collection<T> la plus appropriée pour modéliser les conteneurs.

5) Gestion des fichiers de données

a) Le fichier de données ressources doit contenir des données "réelles" : motos et autos existantes :

- 15 ressources de chaque type (moto, voiture standard, voiture de luxe)
- 50 ressources au total

b) Le fichier de données adhérents doit contenir au minimum 20 emprunteurs.

c) Vous pouvez en option enrichir le projet en créant une classe conteneur pour les locations et gérer un fichier des locations en cours. Il conviendra alors de proposer des recherches dans les informations disponibles. Au premier lancement de l'application, le fichier des locations est à créer, lors des lancements suivants le fichier est à exploiter et à modifier selon les nouvelles locations et les locations arrivées à expiration.

6) Interface console

LOCATION doit proposer des affichages précis et concis.

Toutes les saisies effectuées doivent être sécurisées.

Vous êtes seuls maîtres des informations affichées / à saisir : pensez à l'ergonomie du programme !

Un menu interactif est vivement conseillé.

7) Interface Graphique

Java propose les classes de Swing afin de réaliser des interfaces graphiques évènementielles. Vous réfléchirez à la conception et à l'ergonomie de votre application, puis vous implémenterez la solution retenue en suivant la méthodologie présentée en cours.

IV Travail à effectuer

Vous devrez écrire en Java l'application LOCATION et rendre :

1) Partie application

Une archive comportant :

- tous les codes source en Java sous la forme d'une archive Java NOM1_NOM2_location.jar
- les fichiers de données utilisés par l'application (Emprunteurs, Vehicules et Flotte, et si possible le fichier des locations).
- Les classes de tests doivent également figurer.

Si votre interface graphique est quelque peu évoluée (ce qui n'est pas demandé par défaut), vous veillerez à fournir tous les fichiers nécessaires à un fonctionnement **autonome** de votre application (exemple : fichiers images, sons, vidéos).

2) Partie rapport

Le rapport de projet contiendra, au delà des rubriques habituelles, des captures d'écran, un description du format du fichier des adhérents, les images des histogrammes calculés (optionnel).

V Echéancier

Travaux à réaliser par étapes :

Etape 1

Rédaction des classes Date/Adresse/Emprunteur/Véhicule/Auto/Moto.

Etape 2

Rédaction de la classe **Exemplaire** : un objet de la classe **Véhicule** ne représente pas un véhicule physique, mais un type de véhicule particulier. Chez un loueur, vous pouvez, par exemple, avoir deux Audi A8 : une avec un kilométrage de 12 000 kms, une autre avec un kilométrage de 130 000 kms. Le kilométrage maximum autorisé est 180 000 kms.

Pour stocker cette information, on aura ainsi : un seul objet véhicule (une auto) nommé "Audi A8", mais 2 objets de classe **Exemplaire** : le premier avec un numéro égal à 1 et un kilométrage de 12 000, et une référence vers l'objet véhicule 'Audi A8', le second avec un numéro égal à 2 et un kilométrage de 130 000 et une référence vers l'objet véhicule 'Audi A8'.

Implémentation des 3 conteneurs d'Emprunteurs / de Véhicules / d'Exemplaires : vous devrez utiliser les classes de votre choix de l'API Collection de Java.

Avec ces conteneurs, on doit pouvoir : ajouter un élément, lister tous les éléments du conteneur, rechercher un élément selon un critère (nom d'un emprunteur, marque d'un exemplaire de véhicule).

Etape 3

Rédaction de la classe d'association **Location** avec édition des devis et factures.

Une location associe un emprunteur avec un ou plusieurs exemplaires de véhicules.

Les prix de location sont associés aux objets de la classe **Exemplaire** et de la classe **Vehicule** : ainsi, un **Vehicule** a un prix de location de base (par exemple, une Audi A8 se loue 500 € /jour), prix modulé par le kilométrage de l'**Exemplaire** : on compte 10% de réduction par tranche de 50 000 kms.

Etape 4

Ajout des fichiers de données + tests de l'application.

Etape 5

Interface graphique.

VI Evaluation

Contraintes de développement et d'implémentation :

- Le langage à utiliser est Java 1.7 (1.8 si vous le souhaitez)
- Tous les attributs des classes doivent être déclarés *private* (ou éventuellement *protected* si justifiable par des considérations d'héritage) et *final* dans la mesure du possible.
- Sauf en phase de mise au point, aucune entrée/sortie dans les méthodes autres que celles de lecture/saisie et d'écriture/affichage n'est autorisée en mode console.
- Le code doit être commentée avec les annotations nécessaires à la production d'une Javadoc.
- Ne pas utiliser de générateur d'interfaces graphiques.

Recommandations :

- Tester les fonctionnalités développées au fur et à mesure de leur développement !

- A chaque étape du développement, garder une copie de la version opérationnelle à ce stade.

Équipes projet.

Le projet est à réaliser en binôme d'un même groupe de TP Java, les membres de l'équipe devant être d'un niveau assez comparable en programmation Java. La constitution des binômes devra être validée par votre chargé de TP.

Critères d'évaluation du travail réalisé :

Multicritères : opérationnalité, respect du cahier des charges et des contraintes, ergonomie, fonctionnalités complémentaires, pertinence des choix techniques, qualité de la programmation et du rapport.

Le rendu du projet se fera en 2 deux étapes :

- soutenance lors du dernier TP : 10 points
- rendu du rapport et des sources 3 semaines après le dépôt du code : 5 + 5 points

Bonus:

Un bonus maximum de 5 points sera attribué aux éventuelles fonctionnalités supplémentaires si toutefois elles sont significatives (intérêt, ampleur, niveau, …).

VII Le rendu

Le rendu est sous la forme **NOM1_NOM2.zip**, archive contenant le fichier ***location.jar*** et les ***fichiers de la javadoc*** de l'application Location.

Les projets sont à déposer sur l'espace de rendu dédié, ceux rendus uniquement par mail seront pénalisés. Tout retard sera pénalisé de 2 points par jour de retard. Merci de votre sens des responsabilités.

Afin de bien mettre en évidence les relations entre les classes de votre projet, vous pouvez utiliser un logiciel qui vous générera un diagramme de classes à partir de vos sources.

L'idéal est d'utiliser un logiciel libre permettant la rétro-conception.

StarUML : https://www.projet-plume.org/mots-cles-proposes-par-lauteur/retro-ingenierie

BOUML : https://www.projet-plume.org/fiche/bouml

Un rapport doit fournir votre analyse et votre conception.

Les éléments suivants doivent notamment y figurer :

• une introduction exposant clairement les objectifs, limites, choix du projet

• un mode d'emploi de l'application

• des schémas décrivant l'architecture fonctionnelle

• des schémas décrivant les structures de données utilisées

• une explication en français et/ou pseudo-code et/ou langage d'implémentation choisi des principales fonctions (code à fournir et à commenter)

• une conclusion résumant le travail effectué et ouvrant des perspectives

• une bibliographie

• une table des matières

Les classes conçues doivent être détaillées aussi bien leur interface que leur implémentation.

Votre application doit être testée grâce à un code client assez représentatif.

Un jeu d'essais (code client), le plus exhaustif possible, est donc à fournir.

Une bonne idée est de réaliser le rapport en même temps que le développement.