TP Programmation Objet – Héritage et Exceptions

V. Berry – Polytech Montpellier

Nous considérons ici le cas d'une société immobilière gérant des logements à la location et à la vente. Pour l'implémentation, vous travaillerez dans plusieurs packages détaillés ci-dessous. Pour simplifier, on ne stockera pas les noms des propriétaires des logements.

Important : vous respecterez les nom des entités, des champs et des méthodes indiquées dans l'énoncé. Vous commenterez votre code.

Partie 1 : package entreprise

Sur Moodle vous trouverez une implémentation du package entreprise. Ce package contient :

- une classe permettant de déclarer une société immobilière
- une classe permettant de déclarer des agences associées à une société immobilière
- une classe de test
- Des classes temporaires (à utiliser en attendant les autres packages)

Quand une agence reçoit un nouveau logement à gérer, la business logique implique ce logement est d'abord placé dans une liste des logementsNonGeres, puis plus tard quand un employé de l'agence sera affecté à la gestion de ce logement, le. logement passera dans les logements gérés, stockés dans une Map associant au logement (clef) son gestionnaire (valeur).

1) Le code communiqué a été réalisé par un enseignant peu consciencieux, il a fait des erreurs et des oublis qui font que le code ne compile pas correctement. Pouvez-vous corriger la situation et faire que la classe de test compile et s'exécute correctement sans supprimer les lignes de code qu'elle contient (mais en les corrigeant si nécessaire) ?

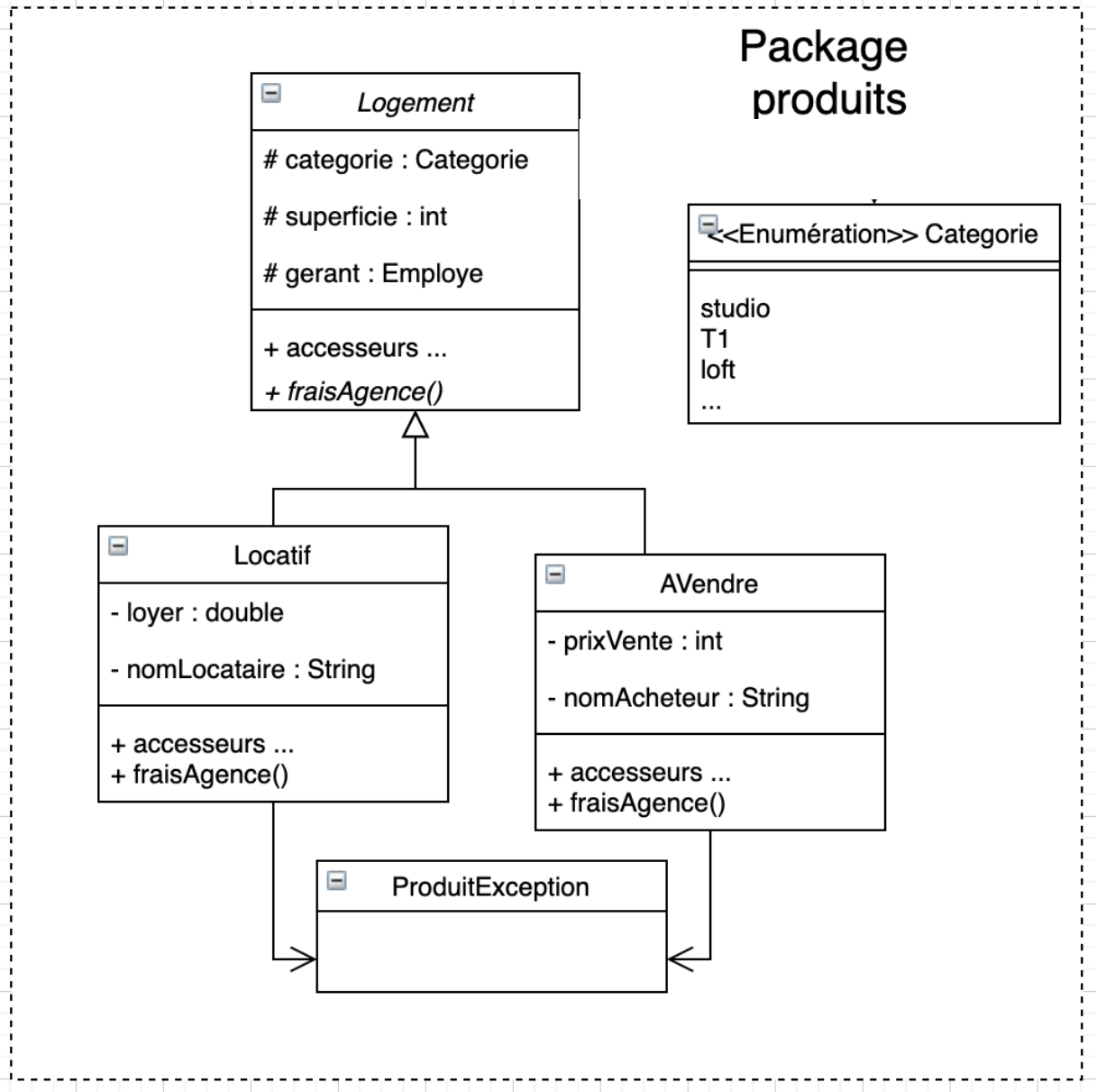
2) Cette classe de test n'est pas très complète, elle ne crée pas du tout de logement. Remédiez à ça en créant dans son main deux LogementTmp, puis associez l'un à une agence et l'autre à une autre agence (utilisez le bon accesseur pour que ces logements aillent se placer dans les logements non gérés (selon le principe évoqué ci-dessus).

3) Remplacez maintenant les affichages « PBM!! » par la levée d'une exception de type Exception (on ne définira pas ici de classe plus spécifique).

4) Dans la classe Societe, indiquez en commentaire dans le code (//TODO…) le pourquoi de la balise @Override au dessus de toString().

C'est le moment de remettre sur Moodle la partie 1 de ce sujet : vous déposerez une archive zip contenant les sources et les binaires du package entreprise **dans son état actuel (incluez les sources <u>et</u> les binaires de <u>toutes</u> les classes).**

Partie 2 : package produits



Le package produits contient des classes et une énumération permettant de modéliser les logements gérés par une Societe. On vous propose le schéma UML de la figure, représentant les entités à déclarer et à implémenter.

1) L'essentiel des méthodes à mettre en place sont des accesseurs, un toString() et un constructeur pour chaque classe concrète (sauf ProduitException). Allez-y c'est à vous d'entrer en action. Note : la classe Employe n'étant toujours pas définie à ce stade, on utilisera EmployeTmp du package précédent comme type pour ce champ présent dans l'entité Logement[1].

2) Le seul point un peu plus complexe ici est la méthode fraisAgence() qui calcule une valeur qu'on ne souhaite pas stocker (c'est ce qu'on appelle une *propriété calculée*). Dans le cas d'un

1 indice: pour utiliser cette classe une clause import est-elle nécessaire, si oui, laquelle ?

logement à la vente, les frais d'agences sont calculés comme étant 5% du prix de vente. Dans le cas d'un logement à louer, les frais d'agence sont un fixe de 20€ + 10€/m^2 de la superficie du logement[2].

3) Pour les logements locatifs et pour les logements à vendre on a en fait besoin de deux constructeurs : un qui renseigne tous les champs (que vous avez normalement déjà écrit), et l'autre qui ne renseigne que la catégorie de logement (il peut arriver que la société rentre un bien sans en avoir encore tous les détails). Ajoutez dans chacune de ces classes un constructeur ne prenant en entrée que la catégorie de logement.

4) Quand on demande de calculer les frais d'Agence d'un logement pour lequel on ne dispose pas des informations suffisantes (superficie pour une location ou prixVente pour un logement à vendre), il faut renvoyer une DetailsProduitManquantsException. Mettez ça en place si vous ne l'avez pas fait précédemment.

5) Dans ce package produits, incluez aussi une classe TestProduits qui teste les différentes classes de ce schéma UML en déclarant au moins un logement à vendre, un autre à louer, et qui fait appel à quelques accesseurs en affichant à l'écran leur résultat. Le logement à louer est créé avec toutes les informations nécessaires, de façon à pouvoir appeler sans soucis fraisAgence(), tandis que pour le logement à vendre vous utiliserez le constructeur n'initialisant pas tous les champs, ce qui conduira à la levée d'une ProduitException. Ces opérations seront effectuées dans sa méthode main(…) de TestProduits.

6) Maintenant que votre package est au point, revenez sur le package entreprise, supprimez la définition de la classe LogementTmp (définie dans TestEntreprise) et remplacez systématiquement son utilisation par la classe Logement du package produits.

C'est le moment de remettre sur Moodle la partie 2 de ce sujet : vous déposerez une archive zip contenant les sources et les .class des packages produits **et** entreprise **(ce dernier ayant été légèrement modifié aussi).**

2 oui, bien sûr normalement il y a aussi les charges, mais n'allez pas vous plaindre quand je simplifie ;-)

Partie 3 : package personnes

Attaquons-nous enfin au package personnes de l'application. Celui-ci doit contenir des entités modélisant l'état et les rôles des personnels d'une société immobilière.

- On stocke le nom et le salaire de tous les employés. Tout employé appartient à l'une de ces trois catégories : le CEO, les partenaires et les agents.
- Le CEO et les partenaires d'une société peuvent gérer le personnel, ce qui consiste à pouvoir exécuter des méthodes recruter(…) et virer(…) :
    - Recruter un employé consiste créer un nouvel employé, renseigner son nom et son salaire, afficher un message de bienvenue et à l'ajouter aux employés d'une agence. Pour l'ajouter à l'Agence, on créera un accesseur ajouterEmp(emp, parQui) dont le 2ème paramètre indique qui fait l'ajout, l'accesseur vérifiera que parQui est le CEO ou un partenaire.
    - Virer un employé d'une agence consiste à le *supprimer* des employes de l'Agence, (créez une accesseur dans Agence avec un paramètre parQui aussi) puis à remettre les logements qu'il gère dans logementNonGeres de l'Agence.
- Le CEO et les partenaires ont une propriété parts (de type float) qui indique le pourcentage d'actions de la société qu'ils détiennent.
- Les partenaires et les agents travaillent dans une Agence (pas le CEO qui gère l'ensemble de la boite). Depuis un tel employé on veut pouvoir connaître cette agence. Les partenaires et les agents ont aussi un portefeuille, c'est-à-dire un ensemble de logements qu'ils gèrent. Ce portefeuille est la vision inverse du champ logementToGerant vu dans la partie 1. D'ailleurs, quand un logement est associé à un employé dans la classe Agence il faut mettre à jour le portefeuille de l'employé en question. De même quand on lui retire la gestion d'un logement, son portefeuille doit être mis à jour.
- Un agent ou un partenaire peut louer(…) un logement à un locataire (conduisant à renseigner le nomLocataire dans l'entité Locatif).
- Un agent ou un partenaire peut aussi vendre(…) un bien, ce qui consiste à renseigner le champ nomAcheteur dans l'entité AVendre de la partie 2. Un agent ne peut vendre un logement qu'au prix de vente prévu, tandis qu'un partenaire peut le vendre à un prix différent de 5% du prix initialement prévu (dans ce cas, le prixVente du logement AVendre est mis à jour). Si le prix de vente convenu ne correspond pas à ces règles, une ProduitException sera levée.

  1) La modélisation de cette situation est plus délicate que les packages précédente. Proposez une modélisation UML des entités intervenant dans le package personnes. Vous la remettez sur Moodle au format **pdf**.

  2) Mettez en place les entités que vous proposez pour le package personnes de façon à assurer la business logique décrite ci-dessus.

  3) Proposez une classe TestPersonnes dans ce package pour tester sa logique interne.

  4) Dans les packages précédents, vous pouvez maintenant remplacer les classes temporaires (dont le nom fini par Tmp) par les classes ajoutées dans ce package. Vérifiez que vous arrivez à compiler ces packages malgré les changements. Vous n'oublierez pas d'ajuster la logique de la classe Agence quand elle demande à un employé de gérer un logement (ou supprime ce lien de gestion) de façon à mettre à jour le portefeuille de l'employé en question.

Remettez sur Moodle la partie 3 : une archive zip contenant les sources et les .class des <u>trois</u> packages (les premiers ayant pu être modifiés en raison des opérations ci-dessus) + votre diagramme UML.

<u>Bonus</u> : produisez aussi un fichier jar *exécutable (et acceptant de s'exécuter) pour les package (merci dans ce cas de proposer un* jar *pour chaque package et d'indiquer dans un* readme.txt *la commande permettant d'exécuter le fichier en question).*