Étude de quelques Design Pattern

Objectif :

L'objectif de ce TP est de comprendre et implémenter différents Patrons de conception en Java.

1 Pattern Observer

Déclarer une interface de type Observer qui a pour seule méthode update(Subject). Implanter une classe abstraite Subject qui contient 3 méthodes : attach(Observer), detach(Observer) et notifyObservers(). La liste des observateurs sera maintenue à l'aide d'une liste (java.util.List).

La boite aux lettres

Notre boite aux lettres va utiliser le pattern Observer pour prévenir ses observateurs d'un changement. Implanter une classe MailBox qui hérite de Subject. Elle comporte les les méthodes read(), newMessage(), getNumberOfMessages() et isEmpty().

Un compteur de mail reçus

Nous allons maintenant implanter un observateur qui va compter le nombre de mails reçus par la boite aux lettres. Déclarer une classe CounterObserver qui implémente l'interface Observer. Afin de pouvoir afficher cet objet, nous allons le faire hériter de JLabel. Il suffit alors d'utiliser la méthode setText() de JLabel pour changer l'affichage de notre Observateur

Représentation graphique de la boite aux lettres

On veut maintenant créer une représentation graphique de notre boîte aux lettres. On utilise pour cela une autre implémentation de l'interface Observer MailObserver qui va afficher le sujet du ou des derniers messages contenus dans la boite aux lettres. Comme pour l'exercice précédent, MailObserver va hériter de javax.swing.JLabel.

2 Étude du patron de conception Visiteur

On veut étudier le patron de conception Visiteur en étudiant un mini-compilateur (voir schéma). L'accent est mis sur les aspects de parcours, avec la présentation (et la réalisation partielle et papier) de deux visiteurs : un pretty-printer et un évaluateur.

Voici un exemple de programme :

```
lire N
```

```
tant_que N > 1 faire
        estImpaire := N % 2
        si estImpaire alors
                N := ( 3 * N ) + 1
        sinon
                N := N / 2
        fin_si
        imprimer N
fin_tant_que
```

Le premier traitement d'un compilateur est la construction d'un arbre syntaxique A partir du programme donné. Les noeuds de cet arbre sont les éléments du langage lui-même.

1) Proposer un diagramme de classes modélisant l'arbre syntaxique du pseudo-langage utilisé.

2) Faire un dessin de l'arbre pour

Block(Assignment(Variable_Ref("x"),Integer_Value(10)),Print(Variable_Ref("x")))

Une partie de la grammaire abstraite du langage est maintenant transmise.

```
Block ::= Statement*
Statement ::= Assignment | Conditional | Print | Read | While | Block
Expression ::= Bin_Expression | Integer_Value | Variable_Ref
Assignment := var: Variable_Ref; rhs: Expression
Conditional ::= cond: Expression; then_part: Block; else_part: Block
Print ::= value: Expression
Read ::= var: Variable_ref
While ::= cond: Expression; body: Block
```

Question : définir les règles *Bin_Expression*, *Integer_Value* et *Variable_Ref* à partir du diagramme de classe.

Question: définir une grammaire concrète pour la grammaire abstraite présentée ci-dessus.

Mise en oeuvre en java de l'arbre abstrait.

Mise en œuvre des visiteurs concrets

1 Écrire le code Java d'un visiteur qui imprime chaque noeud de l'arbre à raison d'une ligne par *Statement*, sans indentation.

2 Ajouter le comptage de la profondeur d'imbrication des blocs.

3 Mettre en oeuvre un interprète *Evaluator* de ce mini langage.

3 Pattern Decorator et State

Énoncé du problème

L'entreprise Gotham Pizzas regroupe une dizaine de petits restaurants appelés "Points Pizza". Cette entreprise est spécialisée dans la livraison à domicile de ses fameuses BatPizzas. Jusqu'à présent les commandes se faisaient par téléphone directement auprès de chaque restaurant. Certains se trouvaient alors surchargés à certains moments, alors que d'autres restaient en sous-régime.

Afin de lisser tout ceci, la direction de Gotham Pizzas souhaite informatiser le processus de commande / fabrication / livraison via un système logiciel baptisé e-Pizza. Grâce à ce logiciel, on souhaite gérer à distance et de manière centralisée l'ensemble des commandes, l'ensemble des Points Pizzas et l'ensemble des employés appelés "Collaborateurs".

A tout moment il est possible de passer une commande par Internet. Le client doit disposer pour cela d'une carte de crédit qui l'identifiera de manière unique. Lors d'une première commande il lui sera également demandé de saisir son nom et de situer son lieu de résidence sur une carte de la ville. Une même commande peut comporter plusieurs pizzas (au plus 5 pour des raisons de transportabilité), et une heure de livraison souhaitée (au moins 30 minutes après le passage de la commande). Chaque pizza est décrite par un nom, mais également par une photo et par un prix. Pour chaque pizza sélectionnée le client doit indiquer la taille désirée (2 ou 4 personnes). La livraison est gratuite à partir de deux pizzas, mais pour une seule pizza elle est facturée 5 euros. Après avoir passé sa commande, le client peut à tout moment consulter l'état de sa commande. Tant que la commande n'est pas encore préparée, il pourra l'annuler s'il le désire. Si la commande est en cours de livraison, il pourra suivre le livreur sur la carte.

Les Points Pizza sont ouverts 24h/24. Pour assurer un service 24h/24 dans toute la ville, Gotham Pizzas fait appel à un très grand nombre de collaborateurs, munis d'un terminal portable intégrant un téléphone portable, et un GPS permettant une localisation suffisamment précise dans la ville. Ces collaborateurs signalent leurs périodes de disponibilité en temps réel par SMS, et sont aussi informés de leurs missions par SMS.

A tout moment le gérant peut consulter l'état du système global. Pour chaque Point Pizza, il peut connaître le nombre de commandes en attente de fabrication, le nombre de commandes fabriquées mais en attente de livraison, et celles en cours de livraison. Il peut affecter un collaborateur soit à un Point Pizza soit à la livraison. En fait un collaborateur peut ainsi changer de lieu de travail ou de rôle plusieurs fois dans une journée: le rôle du gérant est d'optimiser l'attribution de chacun en fonction des commandes. En fait lorsqu'un client passe une commande, il n'indique pas de PointPizza particulier; c'est le gérant qui affecte la commande à un

PointPizza et à un livreur en cherchant en général à optimiser la distance parcourue ainsi que les activités des PointPizzas et des collaborateurs.

Chaque livreur utilise son propre moyen de transport (bus, vélo, roller, voiture, etc.), et informe le système de l'aboutissement d'une livraison. Dans chaque Point Pizza un collaborateur joue le rôle de "coordinateur". C'est le seul à agir directement avec e-Pizza: les autres collaborateurs sont à la préparation des pizzas. Le coordinateur consulte les commandes à réaliser, choisi dans quel ordre les traiter, et a pour charge d'indiquer pour chaque commande lorsque sa préparation débute, lorsqu'elle se termine et lorsque elle est remise au livreur. A ce moment, le livreur reçoit sur son terminal portable les coordonnées du client. Le livreur signale au système que la livraison a eut lieu en envoyant un SMS (prédéfini).

Une première phase d'analyse est effectuée par un collègue, il vous livre alors le diagramme de classe d'analyse présenté en figure 1.

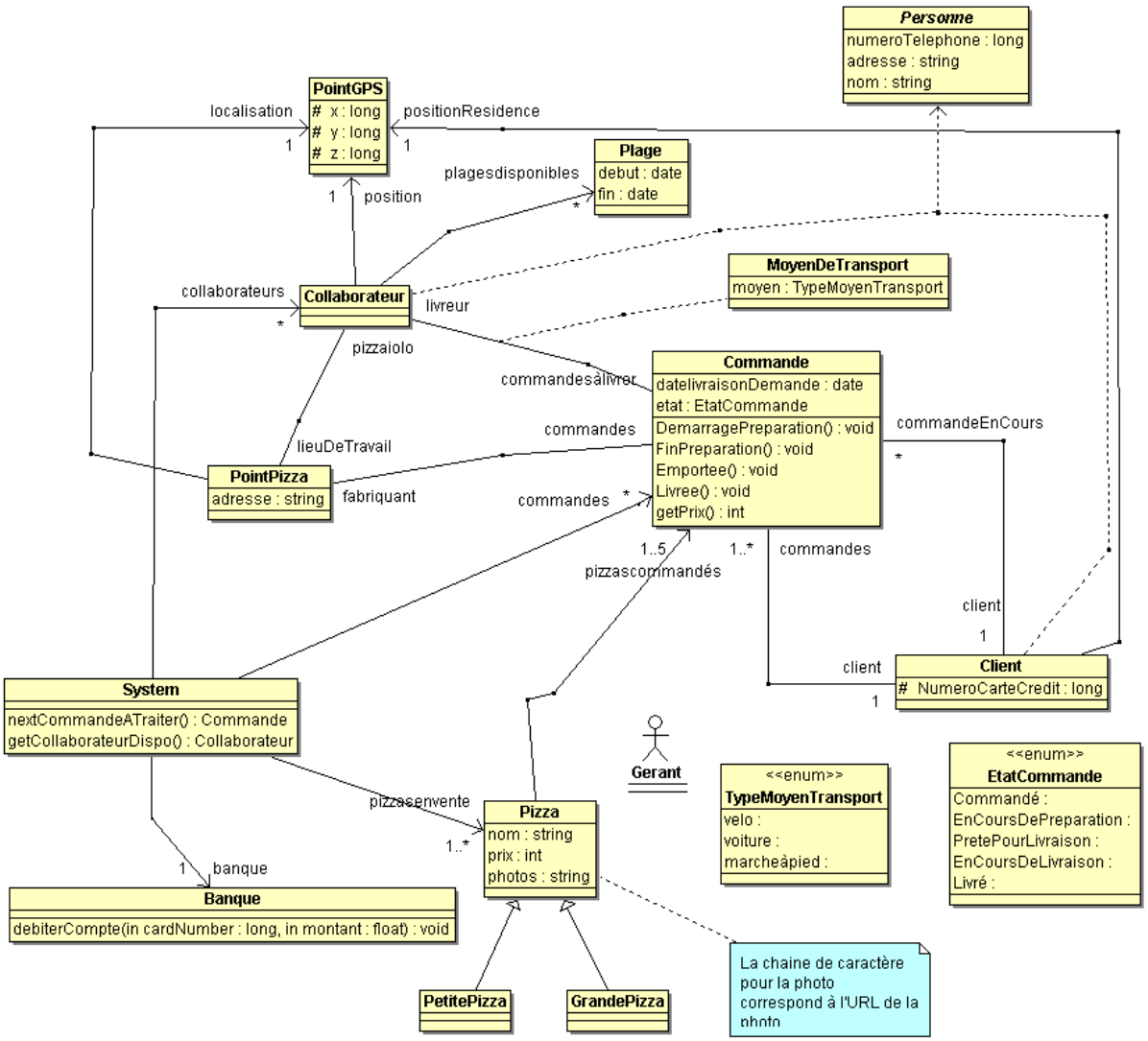

Figure 1: Diagrame d'analyse du système

Question 1 :

Ce diagramme vous semblant (à raison) erroné sur plusieurs points, mettre en œuvre un diagramme de conception en Java pour le système considéré. Dans un Markdown à la racine de votre projet, établissez succinctement les erreurs détectés dans le diagramme de classe d'analyse.

Question 2 :

Suite à de nombreuses plaintes de clients sur les délais de livraison, le responsable de BatPizza se propose d'introduire une nouvelle fonctionnalité à son système. Si la commande est livrée en retard, le client à droit à 10 % de réduction sur son prix par quart d'heure de retard.

Dans ce contexte, il est important de faire attention à une attaque de type *pizza denied of service* afin d'obtenir des pizzas gratuites ou dont le prix diminue trop fortement. En effet, la capacité de production de chaque point de fabrication n'est pas extensible de manière infinie et au delà d'un certain nombre de commande en cours de traitement, la pizza sera nécessairement livrée avec un temps largement supérieur à ½ h.

Pour prendre en compte cette fonctionnalité de traitement de la facturation en cas de retard, on vous propose de mettre en oeuvre le motif de conception (design pattern) Decorator au niveau du calcul du paiement. Définir un diagramme de classe de conception mettant en oeuvre ce motif de conception. Vous pouvez ne reprendre sur ce diagramme que les classes impactées par ce motif de conception.

Dans votre rapport de TP, discutez de l'intérêt de l'utilisation de ce motif de conception. Comparez entre autres par rapport à l'utilisation d'un simple héritage. Discutez aussi par rapport à la mise en place de nouvelles fonctionnalités dans le futur du type *happy hours*[1] entre 15h30 et 17h30.

Question 3 : Pattern State

On souhaite maintenant faire en sorte que certaines opérations dans la classe Pizza ne soient possible que quand la pizza est dans un certain état. On souhaite faire cela en bannissant l'utilisation de la conditionnelle en Java. Proposez une mise en œuvre à l'aide du pattern state. Discutez de l'intérêt de ce pattern.

1 Pizza à 50%.