AJ séance 2

Objectifs :

→ Mettre en œuvre des diagrammes UML fournis.
→ Comprendre le typage fort du Java : les generics et les wrappers classes.
→ Comprendre les différents types de collections (`List`, `Set et Queue`) ainsi que ce à quoi il faut veiller quand on les implémente.
→ Utiliser `Comparator` par le biais d'expressions Lambda.

Thèmes abordés :

- Collections (<p.9 – Queue inclus)
- Generics
- Wrappers
- Classe interne

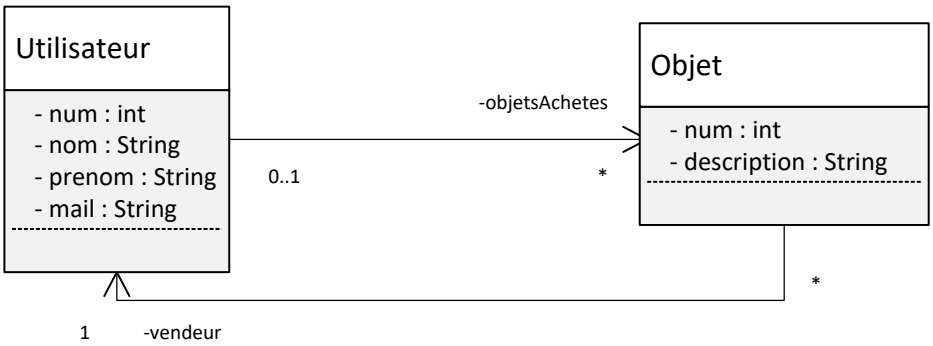
Enoncé

Une équipe d'analystes se penche sur la réalisation d'un site de vente aux enchères. Elle a déjà fourni un diagramme des cas d'utilisation validé par le client :

Les modèles (et scénarios) de ces cas n'ont pas été fournis car l'équipe estime que les intitulés des fonctionnalités du système sont suffisamment explicites.

Dans le système, on vérifie qu'une nouvelle enchère est bien supérieure à la dernière enchère effectuée. Lorsque le vendeur accepte une enchère, l'objet est vendu et n'est donc plus en vente.
Ce qui rend unique chaque objet en vente c'est le numéro qu'il possède. Un utilisateur est identifiable par son numéro également.

Un diagramme des classes conceptuel du domaine a été proposé pour implémenter ces cas d'utilisation.



Montrez dans ce schéma où se trouve les rôles, les sémantique et les multiplicités dans une association.

Analyse

Votre esprit d'analyste averti vous permet de remarquer immédiatement un manquement dans ce diagramme concernant les enchères. Comment sont-elles modélisées dans ce diagramme de classes ?

Soyez également attentif aux **navigations** en veillant aux fonctionnalités spécifiées dans le diagramme des cas d'utilisation.

En plus, nous vous imposons quelques contraintes d'implémentation en Java :

a. Dans l'objet, les enchères sont retenues dans une simple collection**.** Il faut veiller à ce que la dernière enchère soit toujours supérieure et ultérieure à la précédente. (2 versions possibles)
b. L'utilisateur doit retenir les objets qu'il a achetés ; ceux-ci doivent être triés sur base du montant des enchères effectuées, l'objet dont le montant de la dernière enchère est le plus élevé en premier. En cas d'égalité des montants, il faut trier les objets sur leur numéro.
c. L'enchère retient l'objet sur lequel elle est effectuée. Elle reçoit la référence de cet objet à sa construction. C'est également dans le constructeur qu'il faudra garantir la **bidirectionnalité** !

Les attributs `num` présents dans `Utilisateur` et `Objet` sont générés à partir d'une variable de classe privée qui garantit la séquence.

Indiquez tous les types que vous allez utiliser dans votre **diagramme de classes** (qui devient donc un diagramme d'implémentation) et faites-le **valider par un professeur**.

Implémentation

Créez un projet Java dans Eclipse.

Implémentez ces classes dans le package `domaine`.

Pour le moment, il n'y a pas de méthodes, uniquement les getters pour les champs qui ne sont pas des collections.

De même vous définissez un constructeur pour chaque classe. Celui-ci reçoit en paramètres les valeurs des attributs qui ne sont pas multiples (collections). Ces dernières doivent être initialisées à vide. Elles peuvent l'être dans (tous) le(s) constructeur(s) ou, mieux, immédiatement lors de leur déclaration. Le constructeur ne reçoit pas non plus les numéros qui identifient les objets et utilisateurs mais doit leur affecter une valeur séquentielle.

Ces classes représentent la couche domaine de votre application.

Dans la classe `Objet`, implémentez les méthodes suivantes :

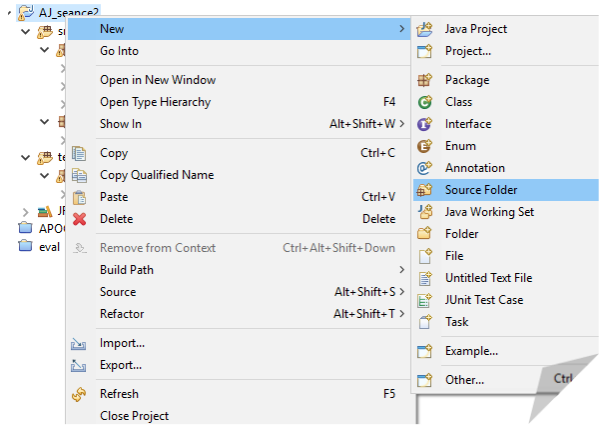
- `ajouterEnchere(enchere : Enchere) : boolean` qui ajoute une nouvelle enchère sur l'objet à condition que le montant soit plus élevé que la dernière enchère effectuée, que la date soit ultérieure et que l'objet ne soit pas déjà vendu. Cette méthode ne doit être visible que par les classes du même package et ses sous-classes.
- `encheres() :List<Enchere>` qui renvoient les enchères dans une collection non modifiable.
- `encheres(date : LocalDate) : List<Enchere>` qui renvoient les enchères effectuées le jour donné en paramètre.
- `meilleureEnchere() : Enchere` qui renvoie la meilleure (dernière) enchère. `null` est renvoyé si aucune enchère n'est présente.
- `estVendu() : boolean` qui précise si l'objet est vendu
- `prixDeVente() : double` qui renvoie 0 si l'objet n'est pas vendu.

Dans la classe `Utilisateur`, implémentez les méthodes suivantes :

- `ajouterObjetAchete(objet :Objet) : boolean` qui rajoute l'objet à la liste des objets achetés s'il ne s'y trouve pas déjà. Il faut vérifier que l'objet n'est pas déjà acheté. Cette méthode doit aussi vérifier que l'utilisateur à qui on ajoute l'objet est bien celui de l'enchère acceptée.
- `objetsAchetes()` qui renvoient les objets achetés par l'utilisateur.
- `objetsAchetes(vendeur : Utilisateur)` qui renvoient les objets achetés par l'utilisateur chez un certain vendeur passé en paramètre

Pour le test des paramètres des méthodes, récupérez l'interface `Util` qui contient les méthodes utilitaires pour effectuer ces tests. Ces méthodes sont implémentées dans l'interface car elles ne dépendent pas de propriétés. Pour les invoquer, importez **adéquatement** l'interface !

Terminez par tester votre application. La classe de tests doit se trouver dans un nouveau source folder du même projet eclipse. Il faut y créer le package correspondant à celui testé, càd domaine :



Si le temps le permet, implémentez la collection d'enchères avec les deux types de collections possibles.