AJ séance 3

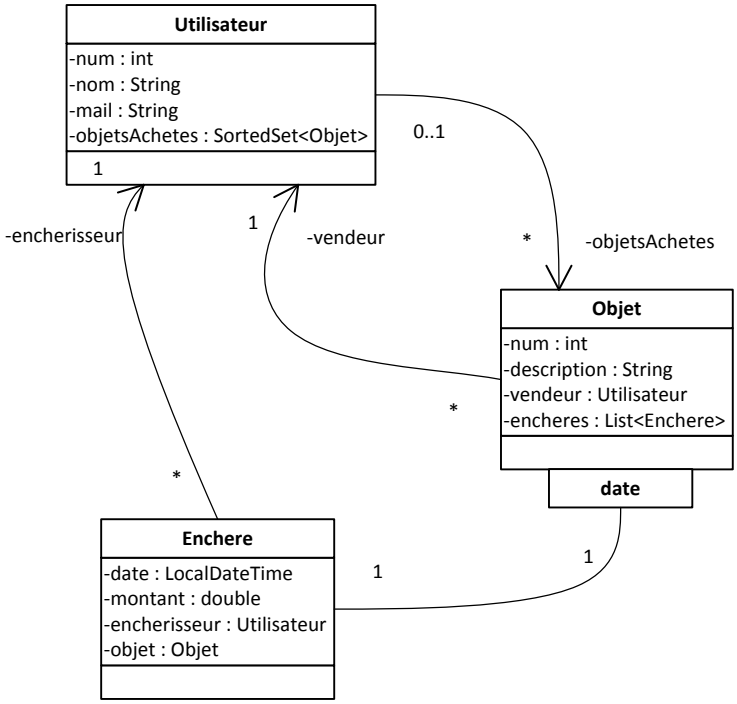
Objectifs :

→ Comprendre et manipuler les `Map`
→ Implémenter une classe comme un singleton
→ Utiliser `Comparator` et exploiter ses possibilités avec des expressions lambda
→ Comprendre l'implémentation d'attributs propres à une classe.
→ Créer des classes d'exceptions adéquates et les utiliser.

Thèmes abordés :

- Attributs
- Collections
- Exceptions
- Singleton (http://thecodersbreakfast.net/index.php?post/2008/02/25/26-de-la-bonne-implementation-du-singleton-en-java)

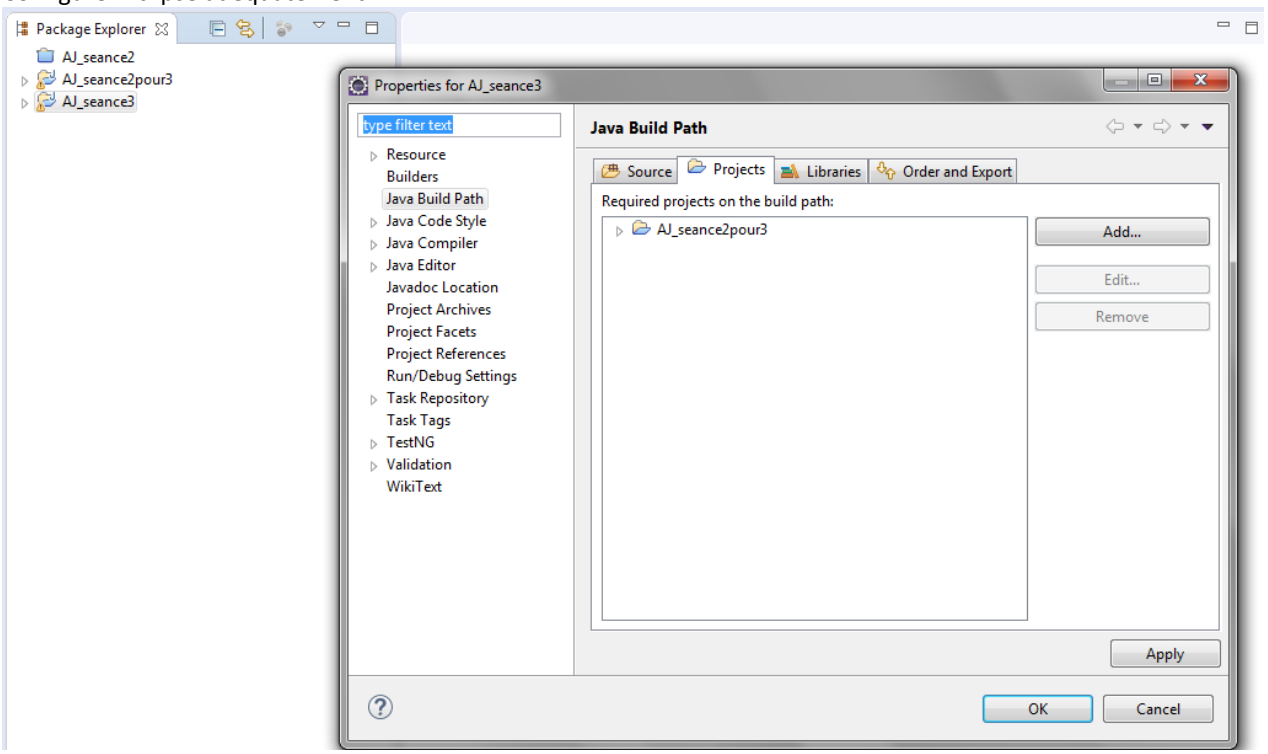
La séance précédente a permis l'implémentation des classes du domaine :



La couche UC représente les différentes fonctionnalités de l'application. Elle manipule les classes de la couche métier afin de répondre aux demandes de l'IHM. La future IHM de votre application ne connaît d'ailleurs que cette couche UC. En bref, l'IHM invoque les méthodes de l'UC qui manipule les objets du domaine.

Il faut dans un premier temps récupérer la solution de la séance précédente qui se trouve dans **seance2** sur claco et l'importer dans votre worskpace. Appelez[1] ce projet **AJ_seance2pour3**.

Créez le package `uc` dans un nouveau projet. Ce projet requiert le projet de la semaine passée dans son *build path.* Configurez Eclipse adéquatement :



L'unique classe du package **`uc`** qui contiendra les fonctionnalités évoquées dans les UC est **`GestionEncheres`**. Elle contiendra 4 collections :

- une **liste** de tous les objets en vente ;
- un **ensemble** des objets vendus ;
- une **map** des utilisateurs mappés par leur numéro ;
- une **map** à définir au point 3.

Les objets sont ajoutés dans la liste des objets en vente par ordre de mise en vente (c.-à-d. par ordre d'appel à la méthode `mettreEnVente`).

1. Pour implémenter le `Set` d'`Objet`, il vous est nécessaire d'implémenter 2 méthodes définies au sein de la classe `Objet`. Lesquelles ?

2. Dans la `Map` des utilisateurs, la clé est un `Integer` alors que le numéro de l'utilisateur est un `int`. Quelles différences y a-t-il entre ces types ?

[1] Par refactoring d'Eclipse

3. Afin de répondre aux besoins fonctionnels, on vous demande d'ajouter encore une collection. Celle-ci fournit, pour chaque jour, les enchères effectuées sur les objets. On veut la liste des enchères par date et puis par le numéro de l'enchérisseur. Pour effectuer cela, ajoutez une Map. Quel va être le type de cette Map ?

Cette classe est implémentée comme un **singleton**.

Elle contient les méthodes suivantes. Ces méthodes testent toutes leurs paramètres en utilisant l'interface `Util` du package `util` **importé** de façon **`static`**.

```
    private Objet rechercherObjet(Objet objet) throws ObjetInexistantException

    private Utilisateur rechercherUtilisateur(Utilisateur utilisateur) throws
UtilisateurInexistantException

    public  Utilisateur inscrire(String nom, String prenom, String mail);

    public  Objet mettreEnVente(String description, Utilisateur utilisateur)
             throws UtilisateurInexistantException;

    public  Enchere encherir(Objet objet, Utilisateur encherisseur,
             double montant, LocalDateTime date)
             throws ObjetInexistantException, UtilisateurInexistantException;

    public  boolean accepter(Objet objet)
             throws ObjetInexistantException, EnchereInexistanteException;

    public  List<Objet> listerObjetsEnVente();

    public  Set<Objet> fournirObjetsVendus();

    public  List<Enchere> listerEncheresDUnObjet(Objet objet)
             throws ObjetInexistantException;

    public  List<Enchere> listerEncheresDUnObjet(Objet objet,
             LocalDate date) throws ObjetInexistantException;

    public  SortedSet<Objet> listerObjetsVendus(Utilisateur acheteur,
             Utilisateur vend) throws UtilisateurInexistantException;

    public  Enchere fournirMeilleureEnchere(Objet objet)
         throws ObjetInexistantException, EnchereInexistanteException;

    public  SortedSet<Enchere> fournirEnchere(LocalDate date);

    public  Set<Utilisateur> fournirEncherisseurDuJour();

    public  Set<Objet> fournirObjetsAchetes(Utilisateur utilisateur)
             throws UtilisateurInexistantException;
```

Quelques explications supplémentaires :

- La méthode rechercherObjet renvoie l'objet enregistré dans la liste et qui correspond à celui passé en paramètre. rechercherObjet lance une `ObjetInexistantException` si l'objet ne se trouve pas dans la liste.

- La méthode `rechercherUtilisateur` effectue le travail similaire pour un `Utilisateur`.

Ces deux méthodes ci-dessus sont des méthodes privées qui sont utilisées dans les autres méthodes afin de ne pas répéter les tests des paramètres reçus (et lancement de l'exception).

Dans les méthodes qui suivent, veillons donc à bien tester les paramètres reçus grâce à ces méthodes.

- La méthode `inscrire` crée un nouvel utilisateur et l'ajoute.

- La méthode `mettreEnVente` crée un nouvel objet et le place dans la liste des objets en vente. Si l'utilisateur n'est pas connu du système, il faut lancer une `UtilisateurInexistantException.`

- La méthode `encherir` ajoute une enchère à l'objet concerné. Elle lance deux exceptions : `ObjetInexistantException` et `UtilisateurInexistantException` selon que l'objet ou l'utilisateur n'est pas connu du système. Avant de créer l'enchère, il faut vérifier que, parmi les enchères du jour, il n'en n'existe pas une effectuée à la même date par le même enchérisseur. Ce test est indispensable pour garantir la cohérence entre la `Map` des enchères qui se trouve dans `GestionEncheres` et la liste d'enchères d'un objet. **Pourquoi ?**

  Indices : La `Map` liste par date tandis que les listes sont par objet…
  La définition du `equals` dans enchère …

  Si l'enchère ne peut être créée car inférieure ou antérieure à la précédente, la méthode renvoie `null`.

- La méthode `accepter` permet au vendeur d'accepter la meilleure enchère pour l'objet concerné. L'utilisateur ayant fait cette enchère possède alors cet objet dans ses objets achetés. L'objet doit être retiré de la liste des objets en vente et ajouté à l'ensemble des objets vendus. Elle lance deux exceptions : `ObjetInexistantException` si l'objet n'est pas connu du système et `EnchereInexistantException` s'il n'existe pas d'enchère pour cet objet.

- La méthode `listerObjetsEnVente` liste les objets encore en vente.

- La méthode `fournirObjetsVendus` fournit l'ensemble des objets vendus.

- La méthode `listerEncheresDUnObjet` renvoie la liste des enchères d'un objet. Elle lance une `ObjetInexistantException.`

- La méthode `listerEncheresDUnObjet` qui prend en plus une date en paramètre renvoie la liste des enchères effectuées un certain jour sur un objet. Elle lance une `ObjetInexistantException.`

- La méthode `listerObjetsVendus` renvoie les objets vendus par un certain vendeur à un certain acheteur. Elle lance une `UtilisateurInexistantException.`

- La méthode `fournirMeilleurEnchere` fournit l'enchère de montant le plus élevé pour l'objet. Elle lance une `ObjetInexistantException`. Elle lance une `EnchereInexistantException` s'il n'y a pas d'enchère.

- La méthode `fournirEnchere` renvoie les enchères d'un jour donné.
- La méthode `fournirEncherisseurDuJour` renvoie les enchérisseurs du jour.
- La méthode `fournirObjetsAchetes` renvoie les objets achetés par un certain utilisateur.

On vous demande :

1) De **créer les exceptions** adéquates. Utilisez l'assistant d'Eclipse.

2) D'implémenter les différentes méthodes de `GestionEncheres`.

3) De faire en sorte que les **attributs de** `GestionEncheres` en soient vraiment. Il faut donc

   a) Il faut donc rendre les classes `Utilisateur` et `Objet Cloneable`.

   Par contre, il sera inutile de faire cela pour `Enchere`. Pourquoi ?

   Mais il faut quand même être attentif à l'attribut enchérisseur d'`Enchere`…

   Dans `Utilisateur` et `Objet`, les collections sont clonées mais par leurs éléments.

   b) Il faut adapter les méthodes `private` de recherche afin qu'elle renvoie le bon objet pour que le travail se fasse sur celui-ci et non sur le paramètre qui est un clone.

   c) Il faut vérifier que les méthodes renvoient des clones ou des objets immuables.

Soyez bien vigilant car cloner de façon générale sans réflexion rendrait votre code inutilisable !