

TP4 de POO : Conception d'un dictionnaire en RMI

Préface :

Ce TP a pour but de continuer les deux précédents afin de constituer un dictionnaire fonctionnant en RMI. De ce fait, les deux classes Dictionary et Factory devront rester sur le serveur, les classes qui sont utilisées devront être Serializable, tandis que l'interface graphique restera sur le poste client.

Rôle des classes :

Pour constituer le dictionnaire, nous avons dû créer deux packages différents : dico.modelisation, et dico.gestionfichier. Le premier nous sert à hiérarchiser une entité (qui contient un mot et une définition, dont le mot contient une catégorie et des propriétés), tandis que le second sert à la création du dico, à sa gestion, mais surtout à sa persistance.

Ainsi, nous avons dans le cas de dico.modelisation :

- ICategory : interface permettant la gestion d'une catégorie
- IProperty : interface permettant la gestion d'une propriété
- IWord : interface permettant la gestion d'un mot
- IEntite : interface permettant les gestions d'une entité.

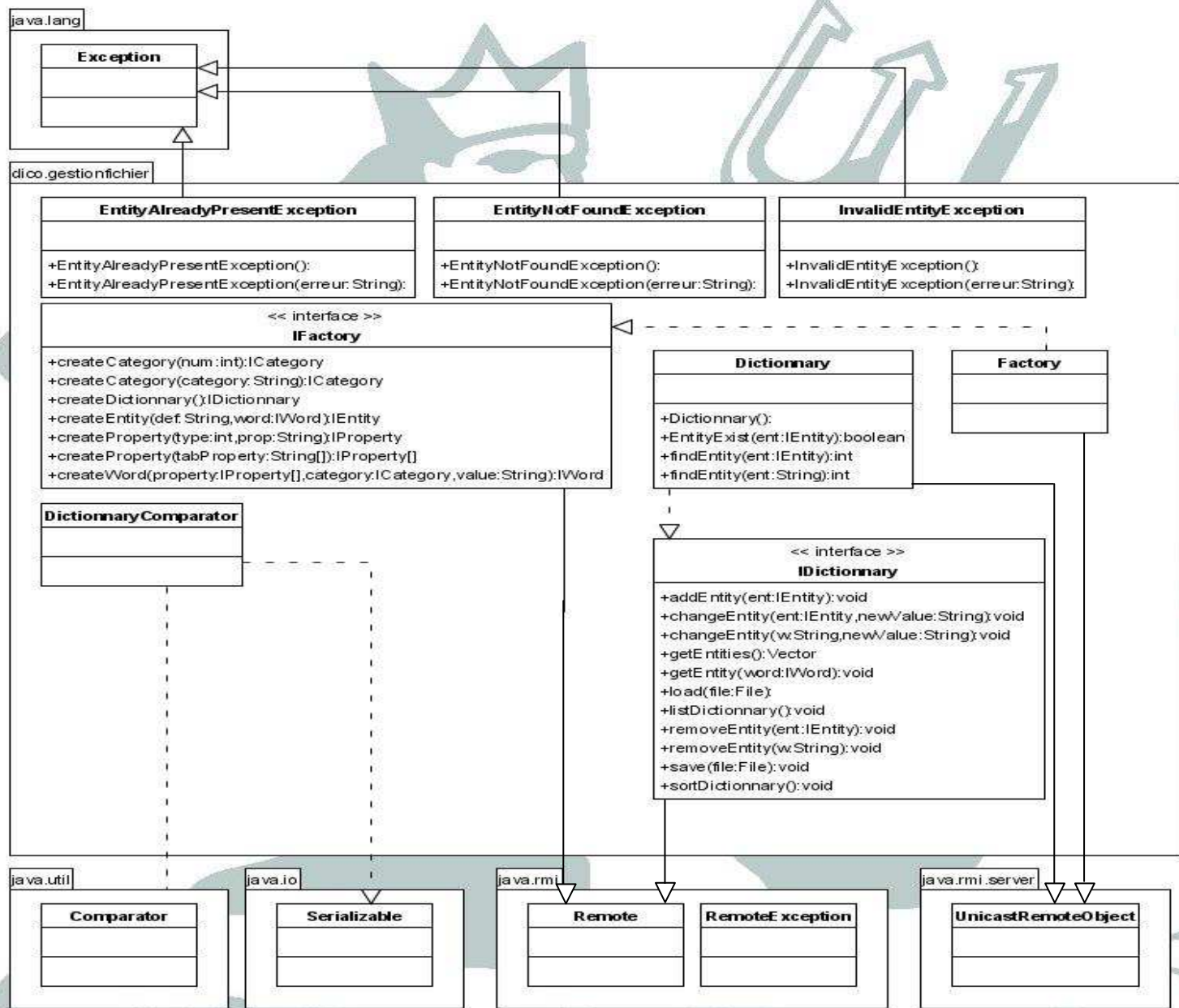
- Category : classe gérant les catégories nécessaires dans notre dictionnaire : adjectif, verbe et nom
- Word : classe gérant le mot en lui-même, mais aussi sa catégorie et ses propriétés
- Entite : classe gérant un mot et sa définition
- Genre : classe désignant une propriété
- Groupe : classe désignant une propriété
- Transitive : classe désignant une propriété
- AbstractString : classe désignant une propriété se gérant par String

- Feminin : classe héritant de AbstractString afin de constituer une nouvelle propriété
- Pluriel : classe héritant de AbstractString afin de constituer une nouvelle propriété

De même avec le package dico.gestionfichier nous avons :

- **Dictionnaire.java** : Cette classe permet de créer un dictionnaire. C'est elle qui possède toutes les méthodes qui permettent de gérer le dictionnaire.
- **DictionnaireComparator.java** : Cette classe sert à trier les mots du dictionnaire. Elle est appelée par la méthode **sortDictionnaire()** de **Dictionnaire.java**
- **EntityAlreadyPresentException.java** : Cette classe est une exception qui est levée lorsque l'on veut rentrer un mot qui existe déjà dans le dictionnaire.
- **EntityNotFoundException.java** : Cette classe est aussi une exception qui est levée lorsque l'on veut supprimer ou modifier un élément du dictionnaire qui n'existe pas.
- **Factory.java** : Cette classe permet de créer une factory. La factory permet de créer un nouveau dictionnaire et toutes les objets qui font parti de celui-ci.
- **IDictionnaire.java** : Cette Interface est l'interface du dictionnaire, la classe **Dictionnaire.java** implémente celle-ci. Cette interface permet de définir les méthodes qui seront utilisées dans **Dictionnaire.java**.
- **IFactory.java** : Cette interface est l'interface de la factory, la classe **Factory.java** implémente celle-ci. Cette interface permet de définir les méthodes qui seront utilisées dans celle-ci.
- **InvalidEntityException.java** : Cette classe est une exception qui est levée lorsque l'on veut créer une entity qui n'est pas formée correctement c'est à dire que les propriétés peuvent être erronées.

Schéma UML de notre dictionnaire en RMI :



Position sur le réseau des différentes classes :

Comme nous l'avons indiqué dans la préface, nous avons mis dans le registre RMI la classe Dictionnaire et Factory. Ainsi, ces deux classes doivent rester sur le serveur. Les classes qui tournent autour de ces deux-là doivent être aussi sur le serveur, mais aussi sur le client du fait que l'interface graphique en utilise pour sa gestion. Ainsi, nous pouvons dire que les classes auront cette position :

Sur le poste Client :

- DictionaryPanel.java (dico.gui)
- DictionaryTableModel.java (dico.gui)
- MainClient.java (dico.gui)
- NewEntityPanel.java (dico.gui)
- EntityAlreadyPresentException.java (dico.gestionfichier)
- EntityNotFoundException.java (dico.gestionfichier)
- InvalidEntityException.java (dico.gestionfichier)
- IDictionnaire.java (dico.gestionfichier)
- IFactory.java (dico.gestionfichier)
- IEntity.java (dico.modelisation)
- IWord.java (dico.modelisation)
- ICategory.java (dico.modelisation)
- Iproperty.java (dico.modelisation)

Sur le poste Serveur :

- MainServeur.java (dico.gui)
- Dictionnaire.java (dico.gestionfichier)
- DictionnaireComparator.java (dico.gestionfichier)
- EntityAlreadyPresentException.java (dico.gestionfichier)
- EntityNotFoundException.java (dico.gestionfichier)
- Factory.java (dico.gestionfichier)
- IDictionnaire.java (dico.gestionfichier)
- IFactory.java (dico.gestionfichier)
- InvalidEntityException.java (dico.gestionfichier)
- AbstractString.java (dico.modelisation)
- Category.java (dico.modelisation)
- Entity.java (dico.modelisation)
- Feminin.java (dico.modelisation)
- Genre.java (dico.modelisation)
- Groupe.java (dico.modelisation)
- ICategory.java (dico.modelisation)
- IEntity.java (dico.modelisation)
- IProperty.java (dico.modelisation)
- IWord.java (dico.modelisation)
- Pluriel.java (dico.modelisation)
- Transitive.java (dico.modelisation)
- Word.java (dico.modelisation)

Du coup, les deux classes qui seront téléchargés seront :

Étapes sur le poste serveur :

Tout d'abord, sur le serveur, nous devons lancer le registre RMI, ensuite nous devons instancier un Dictionary et un Factory, que nous placerons ensuite dans le registre.

Comment le lancer (se positionner dans le répertoire du projet) :

```
Java -jar TestServeur.jar
```

Étapes sur le poste client :

Tout d'abord, nous devons déterminer sur quel registre le poste client doit se connecter. Ensuite, il récupère les interfaces des deux éléments du registre qui seront passés en paramètre sur le programme d'interfaçage.

Comment le lancer (se positionner dans le répertoire du projet) :

```
Java -jar TestClient.jar
```

Conclusion sur le mini-projet :

Ce « mini-projet » a été très instructif pour nous car nous avons pu nous rendre compte que sans organisation, il est très difficile de mener le projet à bien. Pour le passage à RMI, nous avons remarqué l'importance de l'utilisation de classes d'interface. Si un programme est bien structuré il est assez simple de le passer en RMI par contre si on ne prévoit pas ce passage il se peut que l'on ait à reprogrammer des classes entières afin d'arriver à nos fins. Ce mini-projet nous a aussi montré ce qu'était le travail de groupe car chaque binôme devait créer sa partie et la mise en commun est plus compliquée que nous le prévoyions.