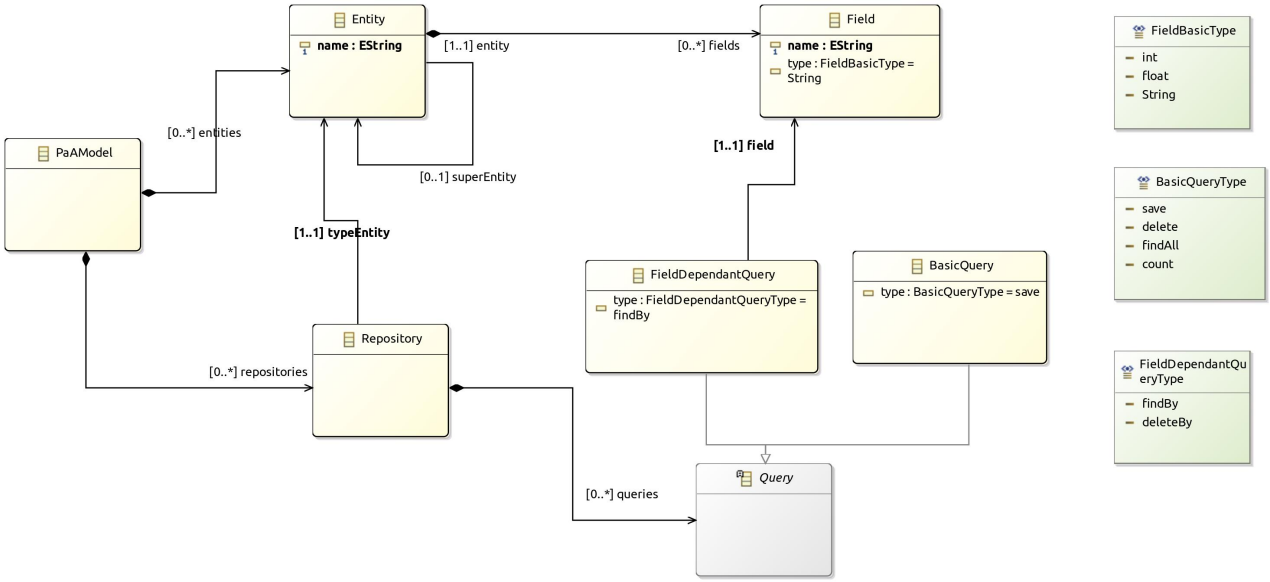
Contrôle continu

Ce travail peut être réalisé en binômes. Barème donné à titre indicatif (sur 21).

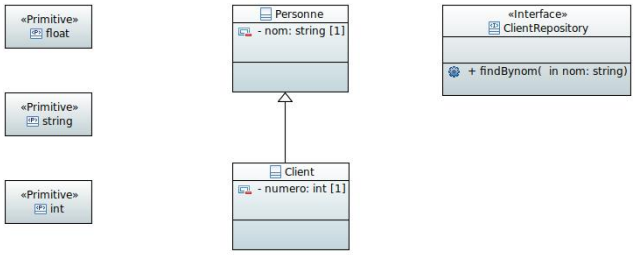
Contexte

On se replace dans le contexte du métamodèle simpliste représentant des entités persistantes et les dépôts associés, avec leurs requêtes, vu lors du CC1 et rappelé ci-après.



Travail à réaliser

Le but est ici de mettre en place quelques éléments d'une transformation d'un modèle d'entités et de dépôts vers un modèle de classes UML pouvant facilement être projeté vers Java/Spring. Par exemple, pour le modèle vu au CC1 et rappelé à la question 2, on veut générer le modèle UML suivant :



Eléments donnés sur moodle

- Le métamodèle PaA en eCore
- Le fichier FileUtils.java contenant une méthode permettant de charger un modèle d'entités persistantes depuis un fichier xmi, et une méthode permettant de sauver un modèle UML dans un fichier (de suffixe uml).
- Un extrait du métamodèle UML indiquant la façon dont les Interfaces contiennent des opérations (référence ownedOperations). Une instance d'interface est aussi une instance de PackageableElement.
- Un extrait du métamodèle UML indiquant la façon dont un élément UML est lié à un commentaire. Les instances de Class et d'Interface ont aussi des instances d'Element.

Question 1. (1 point) Générez le code Java/EMF correspondant à ce métamodèle.

Question 2. (2 points) Saisissez un modèle composé de deux entités nommées respectivement `Personne` et `Client`, avec Client une sous-entité de Personne. Personne a comme champ un nom (type String), et Client a pour champ un numéro (type int). Le modèle se compose également d'un dépôt pour les clients, avec une requête de recherche par nom. Ce modèle est le même que celui vu au CC1.

Question 3. Écrivez en Java/EMF les méthodes suivantes :

1. (1 point) une méthode qui construit et retourne un modèle UML (Model) de nom "generatedModel" et contenant directement les 3 types primitifs int, string et boolean qui seront créés de la manière suivante :

```
private static UMLFactory factory=UMLFactory.eINSTANCE;
private static PrimitiveType uml_int=factory.createPrimitiveType();
private static PrimitiveType uml_string=factory.createPrimitiveType();
private static PrimitiveType uml_float=factory.createPrimitiveType();
static {
uml_int.setName("int");
uml_string.setName("string");
uml_float.setName("float");
}
```

2. (3 points) Une méthode qui prend en paramètre un champ (Field) et qui retourne une property UML privée, de même nom. La property retournée aura pour type l'un des types primitifs décrits précédemment (les types primitifs UML ont les mêmes noms que les types des champs (FieldBasicType)).

3. (3 points) Une méthode qui prend en paramètre une entité (Entity) et qui retourne une classe UML de même nom. Pour chaque champ de l'entité (référence fields) on ajoutera la propriété correspondante (comme vu précédemment) à la classe résultat. Les classes seront commentées avec un commentaire (Comment) dont le body est "@Entity".

4. (4 points) une méthode prenant en paramètre un FieldDependantQuery et retournant une opération publique UML. Cette opération aura pour nom celui du type de requête (findBy ou deleteBy) concaténé au nom du champ (field) référencé par la requête. Cette opération aura un paramètre dont le nom (resp type) est celui du champ référencé par la requête. Concernant le type du paramètre, on réutilisera les types primitifs vus précédemment. Pour simplifier la méthode n'aura pas de type de retour.

5. (4 points) Une méthode qui prend en paramètre un dépôt (Repository) et qui retourne une interface UML ayant pour nom celui du type de l'entité référencée par le dépôt, concaténé à la chaîne "Repository". Pour chaque requête de l'entité (référence queries) on ajoutera la méthode correspondante (comme vu précédemment, et en supposant que toutes les requêtes sont des FieldDependencyQuery, puisque nous ne traitons pas ici les BasicQuery) à la classe résultat. Les classes seront commentées avec un commentaire (Comment) dont le body est "@Repository". Ici il faudrait aussi implémenter Repository¡T¿ où T est le type de l'entité référencée par le dépôt, mais nous ne le ferons pas.

6. (3 points) Une méthode prenant en paramètre un PaAModel et retournant un modèle UML dans lequel on aura placé une classe par entité du modèle d'entité (en générant la classe comme vu précédemment), et en faisant en sorte que si une entité e1 à une super-entité e2, alors la classe générée à partir de e1 a pour superclasse la classe générée à partir de e2. On aura aussi placé dans le modèle une interface par dépôt (Repository) présent dans le PaAModel, en générant l'interface comme vu précédemment.

Qualité de votre code et remise de votre travail

Votre code devra être facilement lisible, et commenté.

Vous indiquerez la procédure pour tester vos transformations de modèles sur le modèle exemple.

Vous remettrez sur moodle une archive de votre projet EMF contenant la réponse aux questions. Si vous avez travaillé à 2, un seul étudiant fait la remise, et précisez bien en commentaire les membres du binôme.