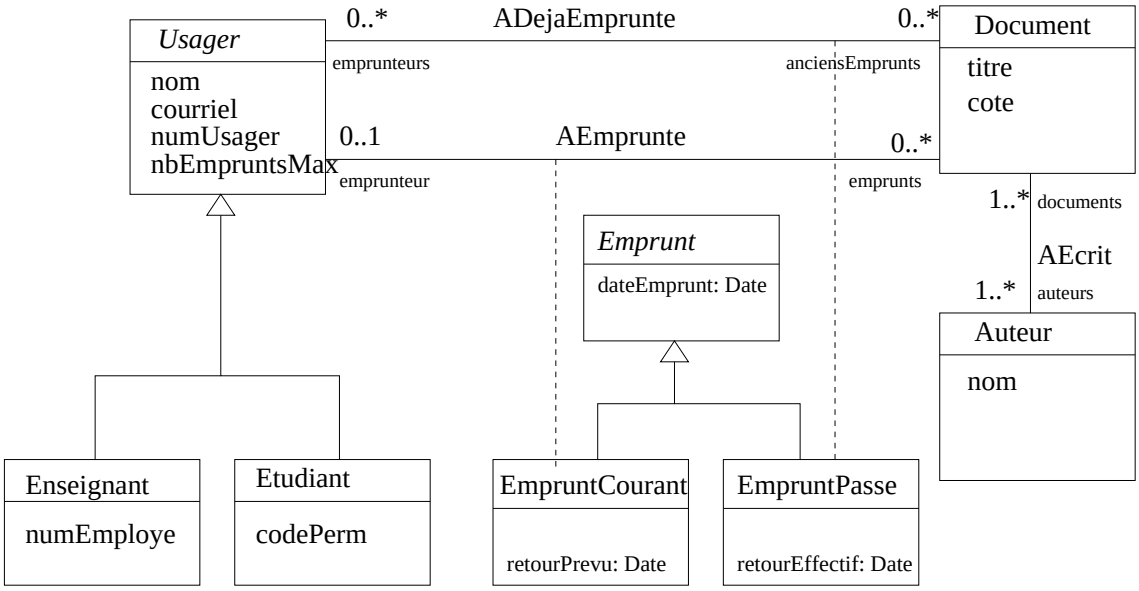
INF3140 : Laboratoire #1

Diagramme de classes

Soit le diagramme de classes suivant, qui modélise des informations pour une bibliothèque universitaire :



Matériel pour le labo

a. Copiez le fichier d'archives suivant dans votre répertoire — notez le «.» à la fin de la ligne :

```
$ cp ~tremblay_gu/Labo1.tar.gz .
```

b. Décompressez ce fichier d'archives — notez le «-» à la fin de la ligne :

```
$  gzcat Labo1.tar.gz | tar xvf -
```

c. Allez dans le répertoire résultant :

```
$ cd Labo1
```

Ce répertoire comprend divers fichiers :

- `bibliotheque.use` : Version textuelle `USE` du diagramme de classe précédent.
- `bib1*.cmd` : Des fichiers de commandes `USE` définissant des systèmes concrets.
- `req*-bib1.cmd` : Des fichiers de commandes `USE` évaluant diverses requêtes sur les objets construits dans le fichier `bib1.cmd`.
- `makefile` : Un fichier pour automatiser certaines commandes (vérifications).

Commandes du `makefile`

- `make modele` : Vérifie la syntaxe du modèle décrit dans `bibliotheque.use` (opérations et contraintes).
- `make objets` : Vérifie que le modèle décrit dans `bibliotheque.use` correspond bien aux objets concrets décrits dans `bib1.cmd` (et vice-versa).
- `make req[abcd]` : Compile le modèle décrit dans `bibliotheque.use`, construit les objets concrets décrits dans `bib1.cmd`, puis évalue diverses requêtes sur ces objets.

  Note : `reqa` est pour la question 1.*a*, `reqb` pour la question 1.*b*, etc.

  Note : Une exécution correcte devrait se terminer en produisant une série de lignes ayant la forme suivante :

  ```
  use> -> true : Boolean
  ```

- `make contraintes` : Compile le modèle décrit dans `bibliotheque.use`, construit les objets décrits dans `bib1.cmd`, puis vérifie que les diverses contraintes (invariants) sont satisfaits — ce qui devrait être le cas si vos contraintes sont correctement définies.
- `make contraintes-pasok` : Idem, mais pour des objets concrets *qui ne satisfont pas les invariants*, donc la vérification devrait échouer pour tous les invariants (`FAILED`).
- `make remise` : Pour remettre votre solution — un simple test pour vérifier que la remise fonctionne bien (comme pour le devoir). Pour cela, il faut modifier le paramètre `CODES_PERMANENTS` dans le fichier `makefile`.

1. Spécification de requêtes (opérations auxiliaires)

Spécifiez, en notation `USE`/OCL, les requêtes suivantes — en respectant de façon exacte le `nomIndiqué` :

a. Une requête `titresDocumentsEmpruntes` qui, pour un usager donné, retourne l'ensemble des titres des documents qu'il a empruntés (emprunts courants) ou qu'il a déjà empruntés (emprunts passés).

b. Une requête `lecteurs` qui, pour un document donné, retourne l'ensemble des noms des usagers qui ont emprunté ce document (emprunts courants ou emprunts passés).

c. Une requête `mesOeuvres` qui, pour un enseignant, retourne l'ensemble des documents dont il est l'un des auteurs.

d. Une requête `mesOeuvresEmprunteesParMoi` qui, pour un enseignant, retourne l'ensemble des documents dont il l'un des auteurs et qu'il a présentement empruntés (emprunts courants).

   Indice : Pour une relation de cardinalité «`0..1`», le rôle associé produit un unique élément, *si le lien est effectivement défini* (et non pas une collection). Dans le cas contraire, le rôle retourne `Undefined`, qu'on peut identifier avec un appel tel que «`r.isUndefined()`».

2. Spécification d'invariants

Spécifiez, en notation USE/OCL, les invariants (contraintes) suivants — en respectant le `NomIndiqué` pour l'invariant (nom qui doit apparaître entre «`inv`» et «`:`» dans la spécification de l'invariant) :

a. `NumeroUsagerUnique` : Le numéro d'usager est un identifiant, donc est unique parmi tous les usagers.

b. `AuteursDistincts` : Deux auteurs distincts d'un même document ne peuvent avoir le même nom.

c. `MaxEmpruntsOK` : Un usager peut emprunter (emprunts courants) au plus le nombre de documents indiqué par l'attribut `nbEmpruntsMax` — cet attribut est fixé au moment de la création de l'usager concret (5 pour un étudiant, 10 pour un enseignant).

d. `MemeTitreAlorsMemesAuteurs` : Si deux document distincts ont le même titre, alors ils ont les mêmes auteurs (copies multiple d'un même document).

e. `CoteUnique` : La cote d'un document est unique.

3. Spécification d'invariants liés aux dates

Pour simplifier la spécification, le modèle `bibliotheque.use` définit une entité `Date` comme n'ayant qu'un unique attribut `val` de type `Integer`. Ceci est suffisant pour modéliser le fait qu'on veuille uniquement définir les prédicats suivants sur une `Date` :

- `egale( autreDate: Date )` : `self` est `egale` à l'`autreDate`.
- `vientAvant( autreDate: Date )` : `self vientAvant` l'`autreDate`.
- `vientApres( autreDate: Date )` : `self vientApres` l'`autreDate`.

En utilisant ces prédicats, spécifiez les invariants suivants :

f. `RetourPrevuVientApresDateEmprunt` : La date prévue de retour d'un document emprunté est nécessairement après la date où l'emprunt a été effectué.

g. `RetourEffectifVientApresDateEmprunt` : La date de retour effectif d'un document ayant été emprunté est nécessairement après la date où le document avait été emprunté.

h. `RetourPrevuVientApresDateRetourEffectif` : La date prévue de retour d'un document emprunté est nécessairement après la date de retour effectif des anciens emprunts.