INF3140–Devoir 3
Automne 2009

À remettre *au plus tard* lundi 14 décembre 2009, à 16h30
Aucun travail ne sera accepté après jeudi 17 décembre 2009, à 15h30
À faire seul ou en équipe d'au plus deux (2) personnes

Remarque préliminaire : Pour ce travail, vous allez à nouveau devoir utiliser l'environnement Unix de l'UQAM (machines `rayon[12].labunix.uqam.ca`), puisque l'outil `USE` (*A UML-based Specification Environment*) est à nouveau requis. C'est aussi par l'intermédiaire de ces machines que vous pourrez utiliser l'outil `oto`, qui sert à la remise (et à la correction) électronique des travaux.

Spécification de contrats en OCL

Présentation du modèle

La figure 1 présente un modèle UML pour des programmes d'études universitaires, où un `Programme` est associé à un `Departement`, contient divers `Cours` (obligatoires ou au choix), et où un `Cours` est caractérisé, entre autres, par un `Sigle` et des `prealables`. Le fichier `universite.use` (qui vous est fourni) contient une représentation dans la notation USE de ce modèle, ainsi que diverses requêtes que vous pouvez — et **devez**, le cas échéant — utiliser.

Opérations à spécifier à l'aide de contrats

Vous devez spécifier les contrats des opérations suivantes (fichier `universite.use`), et ce à l'aide de pré/postconditions. Dans chaque cas, vous devez évidemment vous assurer que **tous les invariants appropriés** sont satisfaits (préservés) — voir le fichier `universite.use` pour la spécification détaillée des invariants.

Remarque : Lorsque j'indique «Étant donné un X», ceci signifie que l'opération demandée doit être définie dans la classe pour X.

a. `ajouterCours( c: Cours, sorte: SorteDeCours )`

   Étant donné un programme d'études, ajoute le nouveau cours indiqué en argument. L'argument `sorte` spécifie si le cours est un cours obligatoire ou un cours au choix : cf. le type `SorteDeCours` (type `enum`) dans le fichier `universite.use`. Le cours ne doit pas déjà être un cours du programme.

b. `supprimerCours( c: Cours )`

   Étant donné un programme d'études, supprime le cours spécifié. Le cours doit évidemment faire partie du programme pour pouvoir être supprimé.

c.
```
creerCours( sigle: Sigle, titre: String,
            numero: Integer, nbCredits: Integer,
            prealables: Set(Cours) ): Cours
```

   Étant donné un département, crée un nouveau cours avec les caractéristiques spécifiées en argument.

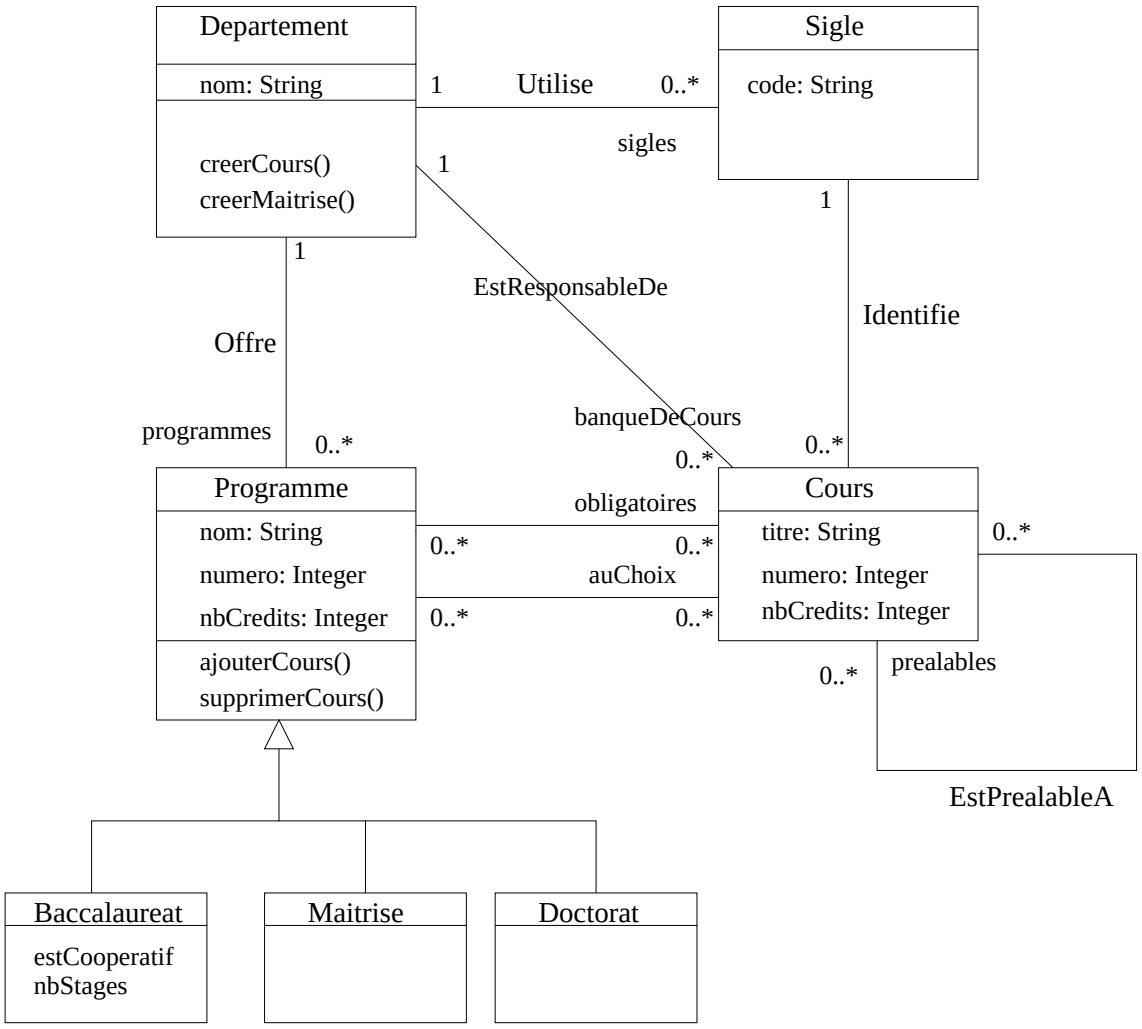

Figure 1: Modèle UML pour des programmes d'études universitaires.

d. `creerMaitrise( nom: String, nbCredits: Integer, obligatoires: Set(Cours), auChoix: Set(Cours) ): Maitrise`

Étant donné un département, crée un nouveau programme de maîtrise avec les caractéristiques indiquées en argument. Le numéro du programme résultant est arbitraire, en autant qu'il respecte les contraintes d'unicité appropriées.

Remarques/Suggestions :

- Vous pouvez introduire des opérations auxiliaires additionnelles — plus spécifiquement, des requêtes. En fait, si (?) certaines expressions *importantes* reviennent fréquemment, vous **devriez** introduire des requêtes auxiliaires — principe DRY = *Don't Repeat Yourself.*
- Pour simplifier les conditions complexes, vous pouvez introduire des identificateurs auxiliaires à l'aide de la construction `let`. Par exemple :

```
nomOperation( a1: T1, ..., an: Tn): R
 pre NomPrecondition:
   let id0: T0 = expr0 in
     ... cond1 ...
 post NomPostCondition:
   let courantes : Sequence(Transaction) = transactionsCourantes@pre in
   let anciennes : Sequence(Transaction) = anciennesTransactions@pre in
     ... cond2 ...
```

- Décomposez les pré/postconditions complexes en plusieurs pré/postconditions indépendantes, et en utilisant pour chaque sous-condition un nom significatif :

```
Contre-exemple:
   nomOperation( a1: T1, ..., an: Tn): R
     post PostConditionComplexe:
       cond1 and cond2 and cond3 and cond4

Exemple:
   nomOperation( a1: T1, ..., an: Tn): R
     post PostConditionA:
       cond1 and cond2
     post PostConditionB:
       cond3
     post PostConditionC:
       cond4
```

- Lorsqu'approprié, utilisez les requêtes (opérations auxiliaires) déjà définies dans le fichier `universite.use`.

Ce qui vous est fourni

Un fichier d'archives (`Universite.tar.gz`) est disponible sur le site Web du cours :
`http://www.info2.uqam.ca/~tremblay/INF3140/Devoirs/Universite.tar.gz`[1]
Ce fichier d'archives contient divers fichiers :

- `universite.use` : Le modèle UML/USE pour les départements, programmes, cours, etc., avec les signatures (seulement) des opérations que vous devez compléter.
- `info.cmd` : Un fichier créant un certain nombre d'entités pour un programme d'études associé au département d'informatique.
- `*.cmd` : Des fichiers de commandes `USE` pour vérifier (tester) les spécifications des diverses opérations.

  Les tests que je vous fournis *ne sont pas exhaustifs.* Entre autres, sauf pour quelques rares cas illustratifs, les tests ne contiennent pas d'appels tels que les préconditions ne sont pas vérifiées ou tels que le résultat produit ne satisfait pas toutes les conditions imposées par les invariarants. Suggestion : Ajoutez vous-mêmes des tests dans les fichiers `X.cmd` appropriés pour tester les diverses préconditions ou postconditions. Les tests, privés, que j'utiliserai pour la correction seront nettement plus exhaustifs quant aux différentes possibilités.
- `makefile` : Un fichier qui permet d'automatiser certaines commandes pour vérifier vos solutions.

Les principales commandes associées à l'utilisation du `makefile` sont les suivantes (commandes exécutées au niveau du *shell* Unix) :

- «`make compile`» : Vérifie la syntaxe du fichier `universite.use` (indépendamment des objets concrets des fichiers `*.cmd`).

  Note : `compile` est la cible par défaut du `makefile`, donc «`make`» (sans argument) équivaut à «`make compile`».
- «`make X`» : Vérifie que le contrat spécifié pour l'opération `X` dans le fichier `universite.use` semble valide — avec les commandes USE du fichier `X.cmd`. Les différentes valeurs pour `X` sont les suivantes : `ajouterCours`, `supprimerCours`, `creerCours`, `creerMaitrise`.
- «`make X VERBEUX=`» : Pour que la vérification de la commande «`X`» se fasse en traçant de façon détaillée et *verbeuse* les diverses commandes exécutées dans le fichier `X.cmd`.

  On peut aussi spécifier «`make X VERBEUX=-q`» qui génère l'exécution la moins verbeuse. La valeur par défaut est «`VERBEUX=-qv`», intermédiaire entre les deux.
- «`make clean`» : Fait le ménage, c'est-à-dire, supprime les fichiers qui peuvent être générés de façon automatique.
- «`make remise`» : Voir plus bas.

[1] Pour décompresser ce fichier d'archives et obtenir les fichiers, il suffit d'exécuter la commande suivante (notez le «-» à la fin de la ligne) :

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

Ce que vous devez remettre

Remise papier

Vous devez remettre une version papier du fichier suivant :

- `universite.use`

Pour la remise de ce document papier, vous devez utiliser **la page couverture (avec grille de correction)** disponible à la page suivante.

Remise électronique

Vous devez aussi remettre une version électronique du fichier précédent, et ce exécutant la commande suivante sur la machine Unix :

```
$ make remise
```

Toutefois, vous devez *auparavant* modifier la valeur de `CODES_PERMANENTS` dans le fichier `makefile` de façon à indiquer vos codes permanent si vous travaillez en équipe (de deux personnes maximum) ou votre code permanent si vous travaillez seul.

Remarques sur la correction :

- Si le fichier remis contient des erreurs de syntaxe, alors vous perdrez une **très (!) grande** partie des points.
- J'effectuerai la correction avec de nombreux autres tests que ceux qui vous sont fournis.
- Une partie des points sera accordée pour le style ⇒ utilisez des expressions les plus simples, claires et explicites possibles.

Travail remis à Guy Tremblay

INF3140-40 : Devoir 3

À remettre au plus tard lundi 14 décembre 2009, 16h30

Ne pas mettre dans une enveloppe

Nom	
Prénom	
Code permanent	
Courriel	

Nom	
Prénom	
Code permanent	
Courriel	

Opération	Syntaxe (1)	Tests publ. (2)	Tests privés (2)	Style (1)	
`ajouterCours`					6 pts
`supprimerCours`					6 pts
`creerCours`					6 pts
`creerMaitrise`					6 pts
Total					24 pts