INF3140 — Modélisation et spécification formelles de logiciels
Examen intra (Automne 2009)

Durée: 13h30 – 16h30 (3 heures) **Documentation autorisée:** Toute documentation personnelle.

Nom: _____

Code permanent: | | | | | | | | | | | | |

Directives:

a. Répondez aux questions directement sur le questionnaire. Pour les questions bonus, vous devez utiliser le verso de la feuille pour votre réponse.

1	/16
2	/12
3	/12
4	/8
5	/18
Bonus	/6
Total	/66

Quelques rappels :

- Soit une collection `col` et une méthode `m` s'appliquant **aux éléments** de cette collection. Un appel de la forme `col.m(...)` (avec «.» plutôt qu'avec «->») est *une abréviation* pour un `collect` :

  ```
  col.m(...) = col->collect( x | x.m(...)  )
  ```

- Soit une séquence `s`. Alors l'ensemble des **index valides** de `s` est l'ensemble `Set{1..s->size()}`.

1. Spécification de propriétés sur des collections OCL (16 pts)

Pour cet exercice, il s'agit de formaliser des énoncés, et ce en utilisant une expression OCL produisant une valeur booléenne. Chaque énoncé décrit une propriété sur une ou sur deux collections. Un exemple vous est donné, servant uniquement *à illustrer* la propriété sur un cas concret — vous devez donc formaliser **l'énoncé général**, portant sur les identificateurs indiqués. Pour votre réponse, vous devez identifier **l'expression ou les expressions** qui modélisent correctement la propriété, et ce **en encerclant la ou les bonnes réponses**.

Voici un exemple :

- Propriété : «Chaque élément de la séquence d'entiers `sq1` a la même parité (pair ou impair) que l'élément correspondant de la séquence `sq2`, et vice-versa.»
- Exemple :

  ```
  sq1: Sequence(Integer) = Sequence{10,20,10,19,33}
  sq2: Sequence(Integer) = Sequence{12,22,14,23,47}
  ```

- Réponses possibles :

  a. `sq1.mod(2) = sq2.mod(2)`

  b. `Set{1..sq1.size()}->forAll( i | sq1->at(i).mod(2) = sq2->at(i).mod(2) )`

  c.
  ```
  sq1->size() = sq2->size()
  and
  not Set{1..sq1.size()}->exists( i | sq1->at(i).mod(2) <> sq2->at(i).mod(2) )
  ```

- Bonnes réponses : a. et c.

a.
- Propriété : «Chaque élément entier du sac `bg` est supérieur à la longueur d'au moins un élément de la séquence `sq`.»
- Exemple :

```
sq: Sequence(String) = Sequence{'affc','a','bc'}
bg: Bag(Integer) = Bag{12,23,32,2}
```

- Réponses possibles :
  (a) `bg->forAll( i | sq->exists( s | i > s.size() ) )`
  (b) `sq->exists( s | bg->forAll( i | i > s.size() ) )`
  (c) `bg->forAll( i | sq->select( s | i > s.size() )->notEmpty() )`
  (d) `sq->select( s | s->exists( i | i > s.size() ) )`

b.
- Propriété : «Les éléments de la séquence `sq` qui apparaissent plusieurs fois dans cette séquence apparaissent aussi plusieurs fois dans le sac `bg`.»
- Exemple :

```
sq: Sequence(Integer) = Sequence{10,20,10,19,44,55,44}
bg: Bag(Integer) = Bag{10,10,10,19,19,44,44,55,55,66}
```

- Réponses possibles :
  (a) `sq->select( x | sq->count(x) > 1 )->forAll( x | bg->count(x) > 1 )`
  (b) `sq->forAll( x | sq->count(x) > 1 implies bg->count(x) > 1 )`
  (c) `sq->forAll( x | sq->count(x) > 1 and bg->count(x) > 1 )`
  (d) `sq->select( x | sq->count(x) > 1 )->reject( x | bg->count(x) > 1 )->notEmpty()`

c.
- Propriété : «La séquence `s` est un *palindrome*, donc se lit de la même façon de gauche à droite que de droite à gauche.»
- Exemple :

```
s: Sequence(Integer) = Sequence{10,20,88,88,20,10}
```

- Réponses possibles :
  (a) `Set{1..s->size()}->forAll( i | s->at(i) = s->at(s->size()-i+1) )`
  (b) `Set{1..(s->size()/2).floor()}->forAll( i | s->at(i) = s->at(s->size()-i+1) )`
  (c) `s->iterate( i; res: Sequence(Integer) = Sequence{} | res->append(i) ) = s`
  (d) `Set{1..s->size()}->collect( i | s->at(s->size()-i+1) ) = s`

d.
- Propriété : «Si on prend n'importe quels deux éléments *distincts* de l'ensemble `st`, ensemble formé d'entiers positifs (> 0), alors le produit de ces deux nombres n'est pas un index valide de la séquence `seq`.»
- Exemple :

```
st: Set(Integer) = Set{10,20..39}
sq: Sequence(Integer) = Sequence{0..11}
```

- Réponses possibles :
  (a) `st->forAll( i1, i2 | i1 * i2 <= sq->size() )`
  (b) `st->forAll( i1, i2 | i1 <> i2 and sq->at( i1 * i2 ).isUndefined() )`
  (c) `st->forAll( i1, i2 | i1 <> i2 implies sq->at( i1 * i2 ).isUndefined() )`
  (d)
```
st->collect( i | st->collect( j | if i <> j then i * j else 0 endif ) )
     ->forAll( k | k > 0 implies k > sq->size() )
```

2. Opérations OCL de base (12 pts)

Soit les valeurs suivantes :

```
sq0 = oclEmpty(Sequence(String))                    -- Sequence{}
sq1 = Sequence{ Sequence{'de',''}, Sequence{'a',''}, sq0->including('') }
st1 = Set{ Bag{-20,10}, Bag{}, Bag{3,4,5}, Bag{0} }
```

Pour chaque expression ci-bas, donnez la valeur résultante dans une forme **la plus simple possible**.

```
Set{1..sq1->size()}->collect( i | sq1->at(i)->size() > sq1->at(i)->first().size() )->size()
=
```

```
sq1->select( s | s->includes('') )->at(2)
=
```

```
sq1->prepend( sq0->union(sq0) )->at(2)->at(1).concat( sq1->at(2)->at(1) )
=
```

```
st1->excluding(Bag{1..0})->excluding(Bag{0})->reject( b | b->size() > 2 )
=
```

```
sq1->select( s | s->size() = 2 )->reject( s | s->exists( c | c.size() > 1 ) )->asSet()
=
```

```
st1->select( b | b->sum().abs() >= 10 )->forAll( b | st1->count(b) > 1 )
=
```

3. Autres opérations OCL (12 pts)

Soit les valeurs suivantes :

```
sb = Set{ Bag{-20,10}, Bag{}, Bag{3,4,5}, Bag{0} }
sq = Sequence{ Bag{'de',''}, Bag{'a',''} }
ss = Set{ Sequence{10,10}, Sequence{3..6}, Sequence{0} }
```

Pour chaque expression ci-bas, donnez la valeur résultante dans une forme **la plus simple possible**.

```
sb->iterate( b; res: Set(Integer) = Set{} |
             res->including(if b->one(x | x > 0) then b->any(x | x > 0) else 0 endif) )
=
```

```
sb->iterate( b; x: Integer = 0 | x.max(b->size()) )
=
```

```
sq->isUnique( b | b->count( 'a' ) )
=
```

```
sq->collect( s | s->size() )
=
```

```
sq->collectNested( s | s.size() )
=
```

```
ss->sortedBy( first() )
=
```

4. Requêtes et contraintes pour des compagnies, employés, véhicules (8 pts)

Soit le diagramme de classes du devoir 1. Complétez les éléments ci-bas, en OCL/USE, en indiquant le nom de classe approprié pour le contexte, le type des arguments, du résultat, etc.

[4] 4.1) Une requête `directeurs` qui, pour une compagnie, retourne l'ensemble des `Employe`s qui dirigent au moins un service de cette compagnie.

```
context                    ::directeurs(                    ):
  =
```

[4] 4.2) Une contrainte (invariant) `BonusDirecteurOK` qui assure que le bonus d'un directeur est supérieur à 5 % de la somme des salaires des employés de chacun des services qu'il dirige — en d'autres mots, pour chaque service qu'il dirige, son bonus est supérieur à la somme des salaires pour les employés de ce service, y compris lui-même.

```
context
  inv BonusDirecteurOK:
```

5. Requêtes et contraintes pour une université (18 pts)

Soit le diagramme de classes présenté à la page 8 — que vous pouvez détacher pour faciliter la lecture — qui modélise des informations sur une université : département, programme d'études, sigle de cours, etc.

[9] 5.1) Spécifiez les requêtes ci-bas. Vous pouvez définir des opérations auxiliaires additionnelles... et vous devriez utiliser les requêtes que vous définissez. Vous devez respecter le `nomDeLaRequete` et utiliser la forme `context NomDeLaClasse::nomDeLaRequete( ... ): ... = ...`

a. Une requête `cours` qui, pour un `Programme`, retourne l'ensemble de tous les cours associés, tant les cours obligatoires que les cours au choix.

b. Une requête `coursGradues` qui, pour un `Departement`, retourne les cours gradués (obligatoires ou au choix) des divers programmes offerts par ce département — un cours est gradué si `numero >= 7000`.

c. Une requête `coursAvecCode` qui, pour un `Departement`, reçoit en argument une chaîne et qui retourne les divers cours offerts par ce département dont le sigle associé possède le `code` indiqué.

[9] 5.2) Spécifiez les contraintes (invariants) indiquées ci-bas. Indiquez explicitement le contexte en utilisant une expression de la forme suivante :

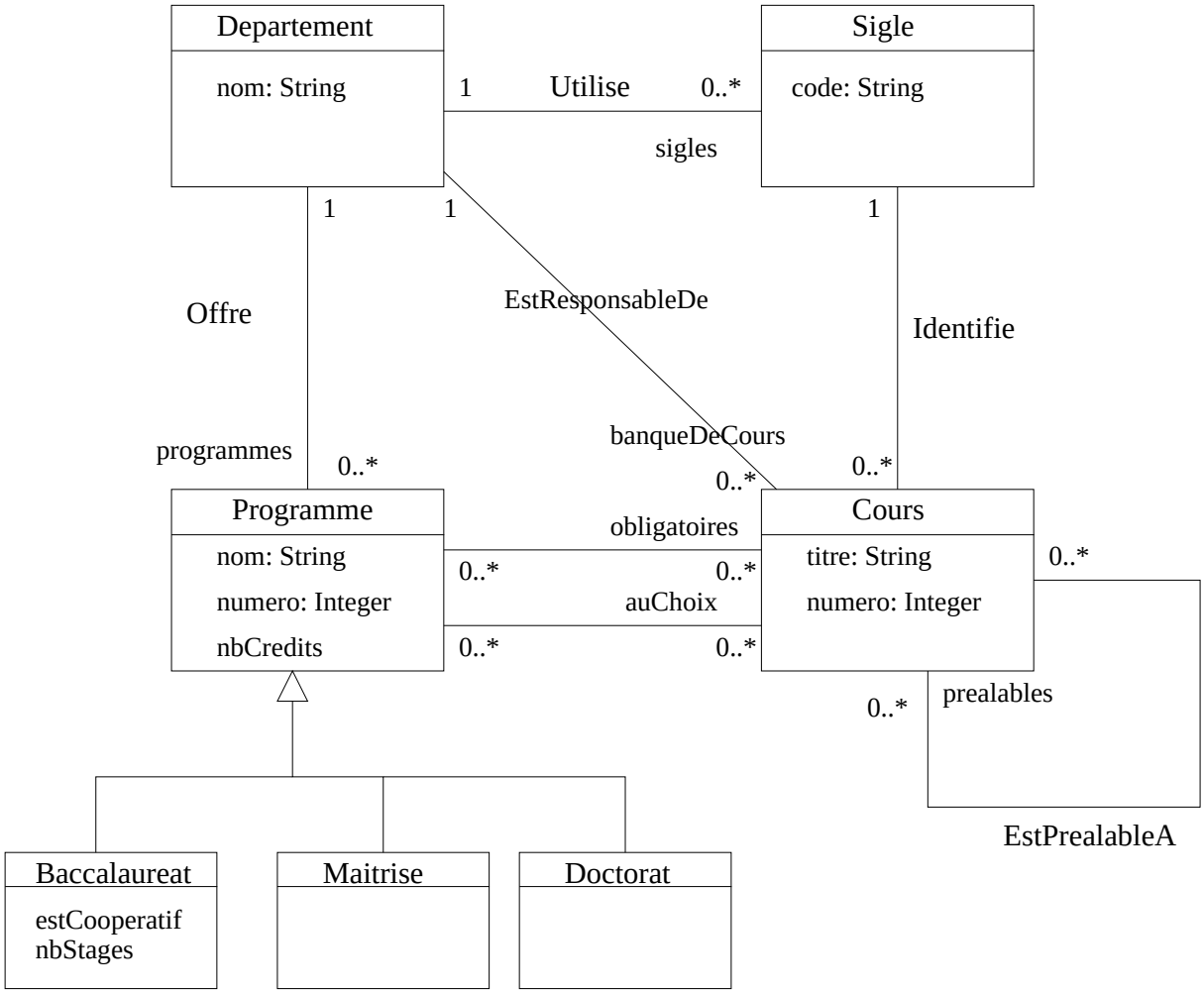
```
context NomDeLaClasse inv NomInvariant:  ...
```

Si désiré, vous pouvez définir des opérations auxiliaires additionnelles ou utiliser les opérations définies précédemment. En fait, lorsqu'approprié, vous **devriez** utiliser les requêtes que vous avez définies.

a. `NumeroProgrammeOk` : Les numéros des divers programmes sont tous des nombres positifs (non nuls) et sont tous distincts les uns des autres.

b. `CoursObligatoiresPasAuChoix` : Pour n'importe quel programme, aucun cours n'est à la fois un cours au choix et un cours obligatoire.

c. `AucunCoursGradue` : Un programmme de `Baccalaureat` ne contient aucun cours gradué — ni comme cours obligatoire, ni comme cours au choix.



Complément d'information sur les `Sigles` de cours :

```
class Sigle
  code: String     -- Par exemple, 'INF', 'MAT', 'ORH', etc.
constraints
  inv CodeLongueur3: code.size() = 3
end
```

Bonus pour question 5 (6 pts)

Pour ces questions bonus, en lien avec la question 5, vous devez utiliser le **verso de la feuille précédente** pour votre réponse.

a. Spécifiez une requête `tousLesPrealables()` qui, pour un cours donné, retourne l'ensemble des cours qui lui sont préalables, directement ou indirectement (préalable de préalable. . .).

b. Spécifiez un invariant `PasPrealableALuiMeme` qui assure qu'un cours n'est jamais préalable à lui-même.