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

Durée: 18h00 – 21h00 (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.

b. **Il n'est pas nécessaire de répéter la signature des opérations ou fonctions que vous devez spécifier.**

c. Utilisez des **noms significatifs** pour les identificateurs de vos pré/post-conditions.

1	/15
2	/5
3	/10
4	/10
5	/10
Bonus	/10
Total	/50

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 en fait *une abréviation* pour une opération `collect` :

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

- `col->sum()` retourne la somme des éléments de la collection `col` (qui doivent être des nombres).
- Soit une séquence `s`. Alors l'ensemble des index valides de `s` est donné par «`Set{1..s->size()}`».
- Dans les expressions régulières, «`\s`» indique un espace alors que «`\w`» indique un caractère pour un mot–identificateur (lettre, chiffre ou «`_`»).

1. Spécification de propriétés sur des collections OCL (12 pts)

Pour cet exercice, vous devez formaliser chacun des énoncés présentés plus bas, et ce en utilisant une expression OCL produisant une valeur booléenne.

Plus précisément, chaque énoncé décrit une relation entre deux collections. Un exemple spécifique vous est donné, exemple qui sert simplement *à illustrer* la relation à formaliser — vous devez donc formaliser l'énoncé général (entre les identificateurs indiqués), et non uniquement l'exemple.

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`.»
- Exemple de valeurs :

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

- Réponse : `sq1.mod(2) = sq2.mod(2)`

a.
- Propriété : «La somme des éléments du sac `bg` est supérieure à la longueur de n'importe quel élément de la séquence `sq`.»
- Exemple de valeurs :

```
sq: Sequence(String) = Sequence{'ac', 'acxc', 'cbc'}
bg: Bag(Integer) = Bag{1,2,2,0}
```

- Réponse :

b.
- Propriété : «Tous les éléments de la séquence `sq` qui n'apparaissent qu'une seule fois dans `sq` font partie de l'ensemble `st`.»
- Exemple de valeurs :

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

- Réponse :

c.
- Propriété : «La séquence `sq2` est *l'inverse* de la séquence `sq1`.»
- Exemple de valeurs :

```
sq1: Sequence(Integer) = Sequence{10,20,10,19,33}
sq2: Sequence(Integer) = Sequence{33,19,10,20,10}
```

- Réponse :

d.
- Propriété : «Si on prend n'importe quels deux éléments *distincts* du sac `bg2`, alors leur somme est un des éléments du sac `bg1`.»
- Exemple de valeurs :

```
bg1: Bag(Integer) = Bag{10,10,22,32,40}
bg2: Bag(Integer) = Bag{0,10,10,22}
```

- Réponse :

2. Types et opérations OCL (10 pts)

Soit les valeurs suivantes :

```
s0: Sequence(String) = Sequence{''}->excluding('')
s1: Sequence(Sequence(String)) = Sequence{ Sequence{'bac','def'}, Sequence{'abc',''}, s0 }
b1: Bag(Set(Integer)) = Bag{ Set{-20,10}, Set{0}, Set{3,4,5}, Set{0} }
```

Pour chacune des expressions indiquées plus bas, donnez la valeur résultante.

```
Set{1..s1->size()}->collect( i | s1->at(i)->size() )->sum()
=
```

```
s1->exists( s | s->includes('') )
=
```

```
s1->excluding(s0)->including(s0)->at(2)->at(2)
=
```

```
b1->forAll( e | e->size() >= b1->count(Set{0}) - b1->asSet()->count(Set{0}) )
=
```

```
b1->excluding(Set{0})->including(Set{0})->select( x | x->sum() > 0 ).size()
=
```

3. Expressions régulières (12 pts)

[9] 3.1) Voici le contenu d'un fichier — les numéros font partie des lignes (font partie du contenu du fichier) mais servent aussi à identifier chacune des lignes dans les questions qui suivent :

```
1. Voici un bout de texte avec des mots aleatoires.
2. Oui, ce texte contient des choses qui n'ont pas de sens et qui sont erronnes.
3. De meme, il contient un ou des chiffres a la premiere position.
4. Ces nombres sont presents ou
5. pas
6. Voici une series de mots sans sens et sans ordre particulier
7. aa ab ac ad ae af
```

Quelle-s ligne-s et morceau-x de texte sera-seront *matchée-s* par les expressions régulières suivantes — plus spécifiquement, si on les utilisait comme arguments pour le programme `Grep.java`.

Vous devez indiquer le-s numéro-s de ces lignes ainsi que le bout de texte qui sera *matché* par le motif — uniquement *la* $1^{ière}$ *occurrence*, puisque `Grep.java` termine l'analyse d'une ligne aussitôt qu'une telle occurrence est trouvée.

a. `"\s+([a-z]+)\1"`

b. `"[^1-467][.].*"`

c. `"\w{4}[0-9]*\w{5}.*"`

[3] 3.2) Écrivez une expression régulière qui permettrait de trouver toutes les lignes d'un texte qui contiennent des mots, composés d'un ou plusieurs caractères et formés d'au moins *deux voyelles consécutives* (majuscules seulement), mots qui sont précédés d'au moins un espace devant et qui ne sont ensuite jamais suivis de chiffres (n'importe où après sur la ligne).

4. Automates sur des bits (8 pts)

Les automates finis sur des bits sont des automates dont l'alphabet est l'ensemble de symboles $\{0, 1\}$. Les mots acceptés par de tels automates sont donc des *chaînes de bits*.

[4] 4.1) Dessinez un automate fini sur des bits (déterministe ou non) qui accepte les chaînes de bits qui *contiennent un nombre impair de bits qui sont à 1* (n'importe où dans la chaîne de bits).

Exemples de chaînes de bits acceptées : «`1`», «`100`», «`0010011`», «`1111111`», etc.

[4] 4.2) Dessinez un automate fini sur des bits (déterministe ou non) qui accepte les chaînes de bits qui *représentent un nombre impair* et qui *contiennent au moins deux (2) autres bits qui sont à 1* (n'importe où *avant* le dernier bit).

Exemples de chaînes de bits acceptées : «`111`», «`10100001`», «`0010011`», «`111111`», «`1000101001`», etc.

5. Requêtes et contraintes pour des compagnies, employés, véhicules (18 pts)

Soit le diagramme de classes présenté sur la dernière page — que vous pouvez détacher pour faciliter la consultation — qui modélise des informations sur des compagnies, services, employés et véhicules.

[9] 5.1) Spécifiez dans la notation OCL/USE les requêtes (opérations auxiliaires) indiquées ci-bas. Vous devez respecter le `nomDeLaRequete` et vous devez spécifier le contexte en utilisant la forme suivante :

```
NomDeLaClasse::nomDeLaRequete(...): ... = ...
```

a. Une requête `conducteursDeTousVehicules` qui, pour une compagnie, retourne l'ensemble des conducteurs qui peuvent utiliser tous les véhicules que possède la compagnie.

b. Une requête `vehiculesDeCompagnie` qui, pour un employé, retourne tous les véhicules de la compagnie (spécifiée en argument) que l'employé peut utiliser.

c. Une requête `dansUnMemeService` qui, pour un employé donné, détermine si un autre employé (spécifié en argument) travaille dans au moins un même service.

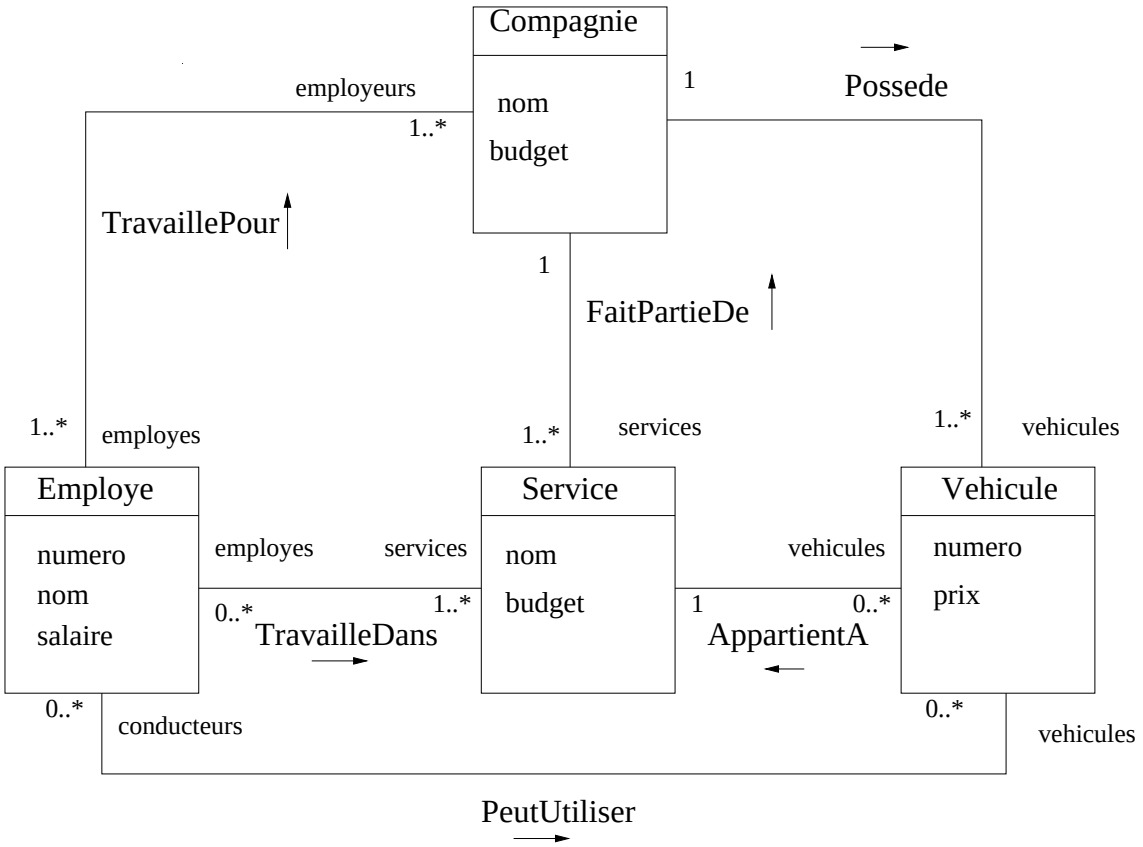
[9] 5.2) Spécifiez les contraintes (invariants) indiquées ci-bas. Indiquez explicitement le contexte en utilisant une expression de la forme suivante :

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

a. `NumeroUnique` : Le numéro (type `Integer`) est unique pour chaque véhicule.

b. `ServicesValides` : Pour n'importe quel employé, les services dans lesquels il travaille sont nécessairement des services qui font partie de compagnies pour lesquelles l'employé travaille.

c. `PrixDesVehiculesOK` : Pour n'importe quel service, la somme totale des prix des véhicules qui appartiennent à ce service ne dépasse pas 10 % du budget du service.



Bonus pour question 5 (7 pts)

Pour ces questions, en lien avec le modèle conceptuel ci-haut, vous devez utiliser le **verso de la feuille précédente**.

a. Spécifiez une requête `employesTresBienPayes` qui, pour une compagnie, retourne tous les employés de la compagnie dont le salaire dépasse 10 % du budget de n'importe quel service dans lequel travaille ces employés.

b. Spécifiez diverses contraintes *additionnelles* sur les entités ou relations du diagramme de classes ci-haut qu'il serait *raisonnable*, et approprié, d'imposer pour assurer la validité et la cohérence du modèle (entre autres, cohérence entre les diverses relations).