MGL7460 — Réalisation et maintenance de logiciels
Examen final (Automne 2015)

Durée: (3 heures) **Documentation autorisée:** Toute documentation personnelle.

Nom: _____

Remarques

- **Vous devez répondre dans le cahier d'examen** qui vous est fourni.
- Répondez à chacune des **questions principales** (1., 2., 3., 4. ou 5.) **sur une nouvelle page**.
- Lorsqu'il est demandé de **justifier** votre réponse, tant la clarté, la justesse que la pertinence des explications seront évaluées — donc si vous écrivez des choses *fausses* ou *«qui n'ont pas rapport»*, vous serez pénalisés.
- Lorsque cela est indiqué, respectez la quantité **maximale** de texte pour votre réponse.
- Pour les deux dernières questions (4. et 5.), vous aurez à écrire du code. Pour ce faire, vous pouvez utiliser le langage de votre choix, y compris du pseudocode. Si vous utilisez un autre langage que Ruby, **vous n'avez pas** à redéfinir les classes et méthodes fournies (celles qui vous ont été transmises par courriel en début de semaine) ; vous pouvez simplement supposer que des classes avec des attributs et méthodes équivalents sont disponibles et déjà définies.

1. Thèmes variés (10 pts)

Répondez à trois (3) questions parmi les cinq suivantes. Vous pouvez aussi répondre à une quatrième question et possiblement obtenir des points bonus. Par contre, je ne corrigerai pas cinq réponses, **au plus quatre**, donc inutile d'en donner cinq.

[3.5] a) Vous venez tout juste de cloner le dépôt `git` d'un-e coéquipier-e. Voici l'état de la copie clonée **immédiatement après l'opération de clonage** :

```
$ ls MCellule
DejaPleineException.class  MCellule.class  MCellule.java
MCelluleMonitor.class  MCelluleMonitor.java  TesterMCellule.java
makefile  makefile~
```

Est-ce correct et acceptable comme dépôt? Que pourriez (ou devriez) vous faire?

Note : Votre collègue utilise `emacs` comme éditeur de texte : si on modifie un fichier «`foo`» avec `emacs`, la version *précédente* du fichier est mise dans le fichier «`foo~`».

[3.5] b) Selon M.C. Feathers,[1] les deux principales qualités des bons tests unitaires sont :

- *They run fast.*
- *They help us localize problems.*

Comment peut-on s'assurer que les tests s'exécutent rapidement si on doit tester des classes qui font des accès à une base de données, à des fichiers ou à un réseau?

[3.5] c) Un cas de test unitaire comporte généralement quatre phases :

Setup	On crée des objets
Exercise	On appelle la méthode à tester
Verify	On vérifie que le résultat ou l'effet produit est celui désiré
Teardown	On nettoie

Il existe deux principales façons — deux principaux styles — pour exprimer l'étape de vérification. Décrivez brièvement quels sont ces deux styles et ce qui les distingue?

[3.5] d) R.C. Martin affirme que «*Tests eliminate fear*». Expliquez/discutez brièvement.

[3.5] e) Discutez l'affirmation suivante : Java étant un langage fortement typé et à typage statique (à la compilation), un **DSL interne** ne peut pas utiliser de techniques de métaprogrammation et doit donc nécessairement utiliser du chainage de méthodes.

---

[1] «*Working Effectively with Legacy Code*», 2005

2. Intégration continue (10 pts)

[8] a) En quoi consiste **l'intégration continue** (*Continuous Integration*) et pourquoi utilise-t-on cette pratique? **(*Max.* $\frac{1}{2}$ *page*)**

[2] b) Dans les années 90, Microsoft utilisait la pratique du «*Daily build and smoke test*» :

> *A common practice at Microsoft and some other shrink-wrap software companies is the "daily build and smoke test" process. Every file is compiled, linked, and combined into an executable program every day, and the program is then put through a "smoke test," a relatively simple check to see whether the product "smokes" when it runs.*
>
> S. McConnell, IEEE Software, vol. 13, no. 4, July 1996.

Expliquez brièvement ce qui différencie la pratique de **l'intégration continue**...

i. de la pratique du «***Daily build and smoke test***» ;

ii. de la pratique de la **livraison continue** (*Continuous Delivery*);

3. Pratiques professionnelles de développement de logiciels (10 pts)

Pour cette question, nous allons supposer que vous êtes un-e étudiant-e inscrit-e à la **maîtrise en génie logiciel** (MGL) de l'UQAM.

Dans le cadre de ce programme, vous devez réaliser un «Projet de synthèse en génie logiciel» de 15 crédits (≈ 500–600 heures de travail).

Un des objectifs de ce projet est de «permettre à l'étudiant-e de mettre en application un ou des aspects de la pratique de pointe en génie logiciel.» Le projet peut être soit un «Projet d'intervention en entreprise», soit un «Projet de recherche appliquée».

Vous avez choisi de faire un «Projet de recherche appliquée». Après discussion avec votre directeur de recherche, vous décidez que vous aller développer une application logicielle utilisable avec une interface Web, application qui sera utilisée et maintenue par la suite par d'autres étudiants.

Vous devez maintenant préparer votre «Proposition de projet», laquelle doit ensuite être approuvée par la directrice du programme de MGL. Votre proposition de projet doit montrer à la directrice que vous connaissez, et saurez utiliser, des pratiques professionnelles de développement de logiciel dans la réalisation de votre projet.

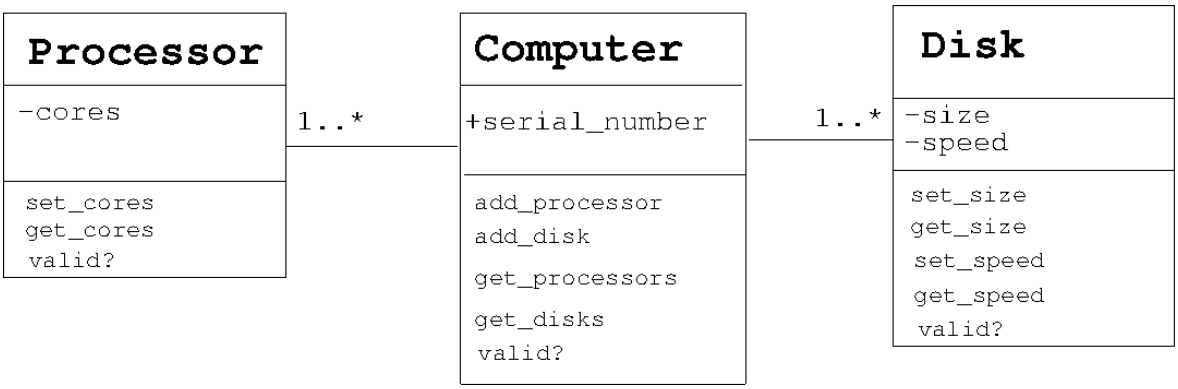
Plus précisément, selon le guide remis aux étudiant-e-s, une telle proposition de «Projet de recherche appliquée» doit contenir les élements suivants :

1. Introduction;
2. Description de la problématique étudiée;
3. Buts et objectifs du projet;
4. **Moyens retenus afin d'atteindre les objectifs prévus en précisant les outils et les méthodes utilisés**;
5. Échéancier de réalisation des travaux;
6. Description des livrables prévus et résultats attendus;

Quels seraient les principaux éléments que vous indiqueriez pour la partie **4.** (Moyens, outils, méthodes)?

(*Max.* 1 *page*)

Note : Les deux questions qui suivent portent sur le modèle UML ci-bas, pour lequel des classes (en Ruby) réalisant ce modèle sont définies plus bas (Programmes Ruby 1 et 2) ainsi qu'un petit programme de test (Prog. Ruby 3).




```ruby
class Processor
  def initialize
    @cores = nil
  end

  def set_cores( cores )
    @cores = cores
  end

  def get_cores
    @cores
  end

  def valid?
    @cores && @cores >= 1
  end
end
```

```ruby
class Disk
  def initialize
    @size = nil
    @speed = nil
  end

  def set_size( size )
    @size = size
  end

  def get_size
    @size
  end

  def set_speed( speed )
    @speed = speed
  end

  def get_speed
    @speed
  end

  def valid?
    @size && @size > 0 &&
    @speed && @speed > 0
  end
end
```

Programme Ruby 1: Les classes `Processor` et `Disk`.

```ruby
class Computer
  attr_reader :serial_number

  def initialize( serial_number )
    @serial_number = serial_number
    @processors = []
    @disks = []
  end

  def add_processor( processor )
    @processors << processor
  end

  def add_disk( disk )
    @disks << disk
  end

  def get_processors
    @processors
  end

  def get_disks
    @disks
  end

  def valid?
    !@processors.empty? &&
    @processors.all? { |processor| processor.valid? } &&
    !@disks.empty? &&
    @disks.all? { |d| d.valid? }
  end
end
```

Programme Ruby 2: La classe `Computer`.

```
# Configuration d'un objet Computer
d1 = Disk.new
d1.set_size( 100 )
d1.set_speed( 200 )

p1 = Processor.new
p1.set_cores( 2 )

c888 = Computer.new( 888 )
c888.add_processor( p1 )
c888.add_disk( d1 )


# Verification des proprietes des processeurs.
c888.get_processors.size.must_equal 1
c888.get_processors[0].get_cores.must_equal 2

# Verification des proprietes des disques.
c888.get_disks.size.must_equal 1
c888.get_disks[0].get_size.must_equal 100
c888.get_disks[0].get_speed.must_equal 200

# Verification de la validite.
assert c888.valid?
```

Programme Ruby 3: Un programme de test qui décrit un ordinateur avec une configuration valide.

4. Tests d'acceptation (10 pts)

On veut définir des tests d'acceptation pour l'utilisation de ce modèle et de ces classes. Voici un récit utilisateur et un scénario simple (cas normal) exprimé en `Gherkin`.

```
Feature: Configuration de nouvelles machines

En tant que technicien responsable des nouveaux ordinateurs
Je veux pouvoir configurer les machines
De facon a avoir les configurations requises par leur utilisateurs


Scenario: On configure une machine avec un disque et un processeur

Given un disque "d1" de capacite "100" et de vitesse "200"
Given un processeur "p1" avec "2" coeurs

When on configure la machine "888" avec le processeur "p1"\
                                  et avec le disque "d1"

Then le numero de serie de l'ordinateur est "888"
And l'ordinateur a "1" processeur avec "2" coeurs
And l'ordinateur a "1" disque de capacite "100" et de vitesse "200"
And la configuration de l'ordinateur est valide
```

[2.5] a) Donnez la spécification **d'un scénario** (un seul) qui décrit **une configuration non valide**.

[7.5] b) Soit le scénario présenté dans l'encadré ci-haut qui définit une configuration valide (`On configure une machine avec un disque et un processeur`). Donnez la mise en oeuvre des `steps` suivants (déclarations **et** mises en ouvre des méthodes avec *patterns* appropriés), qui vont permettre de rendre exécutable ce scénario à l'aide d'un outil tel que `jBehave`, `cucumber` ou autre :

i. La deuxième clause `Given` (`Given un processeur "..." avec ...`)

ii. La clause `When` (`When on configure la machine "..." avec ...`)

iii. La première clause `And`[2] (`And l'ordinateur a "..." processeur...`)

[2]Dans cet exemple, les clauses `And` sont équivalentes à des clauses `Then`.

5. Définition d'une API coulante pour un DSL interne (10 pts)

L'API des classes `Processor`, `Disk` et `Computer`, qui vise à configurer des machines, provient d'un système légataire (un «vieux» système, toujours en fonction) et utilise un style **impératif** pas très élégant.

On désire définir une nouvelle API, qui sera dans un style plus **déclaratif** et plus coulant — *a fluent interface* — API qui pourra aussi être utilisée pour définir des scripts de configuration de machines à l'aide d'un DSL interne.

Notamment, la nouvelle API évitera la forme `get_x` pour accéder à un attribut `x`, utilisant plus simplement une méthode `x`. De plus, cette API utilisera du chainage de méthodes, approche pouvant être utilisée dans la plupart des langages de programmation. Votre tâche est de définir et mettre en oeuvre une telle API.

[5] a) Pour bien préciser cette nouvelle API coulante, réécrivez le Programme Ruby 3 (p. 7), tant la partie configuration (3.5 pts) que la partie vérification (1.5 pts). Introduisez un objet `c888` pour pouvoir vérifier ses propriétés. Introduisez aussi des objets auxiliaires `d1` et `p1`, et ce même si ces objets ne sont pas vérifiés directement, et utilisez ces objets dans la construction de l'objet `c888` — l'objectif est que la classe `Disk` soit indépendante et autonome, pour ensuite répondre à la question suivante.

[5] b) Donnez la mise en oeuvre, dans le langage de votre choix (voir remarque p. 1), des nouvelles méthodes **de la classe** `Disk` (et uniquement cette classe) permettant de mettre en oeuvre votre API coulante.