UQAC

Département d'informatique
et de mathématique
Université du Québec à Chicoutimi

Département d'Informatique et de Mathématique (DIM)
Développement avancé de logiciels: patrons et modèle (8INF956)
Hiver 2018

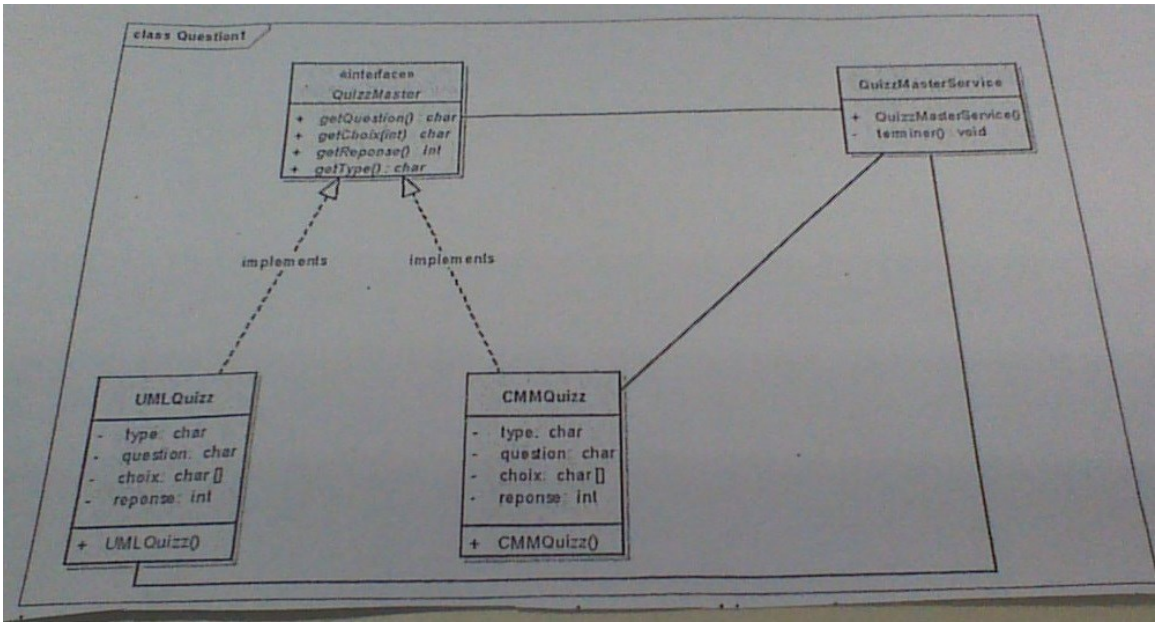
Professeur : Hamid Mcheick	Ce travail est individuel
Semestre : Hiv. 2018	Date de distribution : 16 janvier 2018
Pondération : 15 points	Date de remise : 6 février 2018

Le but du TP est de familiariser l'étudiant avec les concepts suivants:

- L'adaptation d'une application logicielle
- le connecteur et l'interaction entre les composantes logicielles
- le patron de conception et l'injection de dépendance
- la modification de l'architecture pour diminuer le couplage des applications
- les styles/patrons architecturaux, protocoles d'interactions entre composantes
- la séparation des préoccupations: AOP, AspectJ

Question 1 : (45 points)

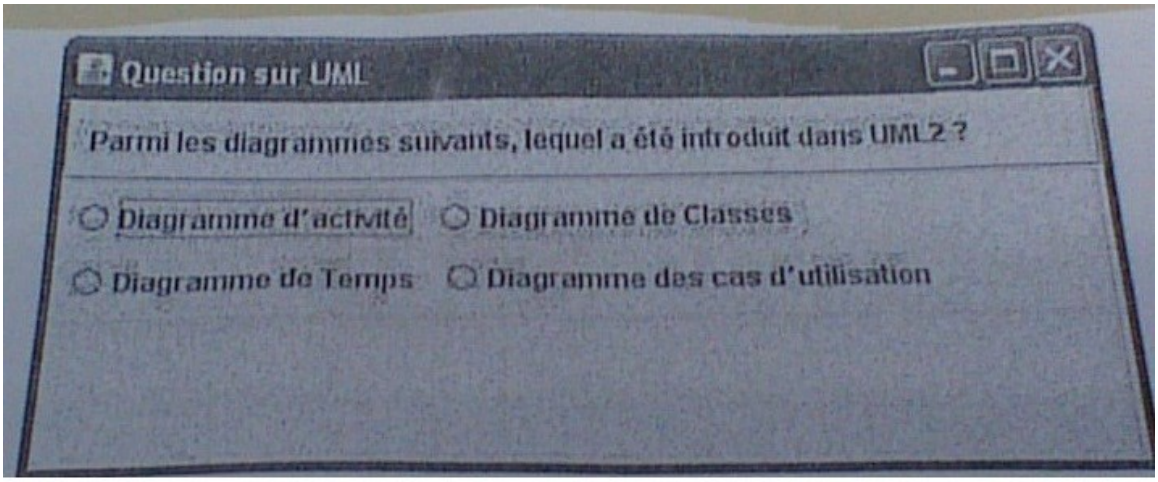
Bill Gate, un ingénieur logiciel, désire réaliser un logiciel de quiz pour un professeur en informatique de l'UQAC. Dans un premier temps, il développe une première version Java du logiciel en se basant sur le diagramme de classes suivant :



Cependant, Bill n'est pas satisfait car il trouve, que dans son architecture, le couplage entre les différents composants est fort, ce qui à la longue pourrait lui compliquer les opérations de maintenance du logiciel. Par exemple, pour créer une instance QuizzMaster dans la classe QuizzMasterService, il a fait :

```
private QuizzMaster quizMaster= new UMLQuizz(); // (ligne UML)
```

L'exécution de l'application donnerait alors comme résultat :



Pour afficher le questionnaire sur le modèle CMM, il faudrait modifier la classe QuizzMasterService de la manière suivante : (remplacer la ligne UML par la ligne CMM)

```
private QuizzMaster quizMaster= new CMMQuizz();// (ligne CMM)
```

Ayant entendu dire que vous étiez des experts Spring, Bill vous demande de modifier son application en utilisant le design pattern « injection de dépendance ».

Vous devez proposer deux nouvelles versions Spring de l'application de Bill, une injectant les dépendances à l'aide de constructeurs et l'autre à l'aide de setters.

Vous trouverez le code source de l'application originale de Bill avec ce travail.

Question 2 : (45 points)

Implémentation du patron architectural publisher-subscriber en utilisant le patron de conception Observer avec la programmation par aspect (AspectJ).

Le patron architectural publisher-subscriber supporte très bien le couplage faible au niveau architecture de logiciel. Nous souhaitons de mettre en œuvre ce patron. Il s'agit de développer un programme d'interaction entre deux ou plusieurs composantes en se basant sur le patron Observer/Observable. Ce programme implémente le protocole d'interaction entre ces composantes et permet de simuler leur relation <dépendants>.

La programmation par aspect, une de méthodes de séparation des préoccupations, permet d'adapter des applications (ou composantes) et de réaliser un protocole d'interaction en module (aspects). La programmation par aspects propose de structurer les programmes en les décomposant en aspects et classes. Un aspect est une unité de décomposition transversale au découpage de l'application en classes. Un aspect peut avoir une ou plusieurs propriétés: points de jonction, introduction des méthodes, des attributs, des classes, etc. AspectJ est une extension de Java pour la programmation par aspect. Vous pouvez utiliser AspectJ avec Eclipse, NetBeans, ou autres outils pour développer votre programme.

Question 3 : (10 points)

Q3.1 Pourquoi les logiciels utilisés en entreprises doivent obligatoirement évoluer ? Énumérer trois raisons et donner un exemple.

Q3.2 Pourquoi les patrons et le modèle (MDA) peuvent être une solution pour remédier à ce problème d'évolution ? Discuter? Énumérer trois avantages de cette solution.