TSE1 – QCM Mini-projet informatique TDA
le 8/06/2012

Rappel de l'énonce du mini-projet

Il s'agit de développer une application permettant de gérer un questionnaire à choix multiples (QCM) sur ordinateur. L'application affiche des questions et propose plusieurs réponses à l'utilisateur qui doit en choisir une ou plusieurs par question. Une fois qu'il a répondu à toutes les questions et qu'il ne veut plus modifier ses réponses, il valide le QCM. L'application écrit alors les réponses dans un fichier et affiche une note. Les questions ainsi que les réponses proposées seront fournies sous la forme d'un fichier texte. Il n'y aura pas de limitation de temps pour répondre aux questions.

L'application devra disposer des fonctionnalités suivantes :

1. Charger les questions du QCM à partir d'un fichier.
2. Gérer le déroulement du QCM en proposant les questions à l'utilisateur et en lui permettant de saisir ses réponses.
3. Permettre à l'utilisateur de revoir n'importe quelle question, de modifier ses réponses puis de valider le QCM (lorsqu'il a validé, il ne peux plus modifier ses réponses)
4. Calculer et afficher la note globale obtenue.
5. Enregistrer dans un fichier les réponses de l'utilisateur ainsi que sa note.

Partie 1 : Spécifications

Les 2 écrans choisis pour interagir avec l'application sont :

- un écran de menu affiché au lancement de l'application qui permet à l'utilisateur de choisir ce qu'il veut faire
- un écran de réponse qui permet de voir les questions et de sélectionner ses réponses.

Dans l'écran de réponse, l'utilisateur peut sélectionner une réponse en tapant la lettre correspondant à la réponse choisie. L'écran est alors réaffiché et la réponse sélectionnée est précédée du symbole « * ». S'il sélectionne à nouveau cette même réponse, elle est désélectionnée et le symbole « * » disparaît. Lorsqu'il navigue d'une question à l'autre en tapant « s » ou « p », les réponses déjà sélectionnées apparaissent précédées du symbole « * » ce qui lui permet de voir les réponses qu'il a déjà saisies pour cette question et éventuellement de les modifier.

Les écrans auront l'allure suivante:

Ecran de menu

```
Application QCM
QCM en cours : QCM-POO.txt

Menu
1) Charger un QCM
2) Répondre aux questions
3) Valider le QCM et calculer la note
4) Enregistrer les réponses dans un fichier
6) Quitter l'application

Entrer votre choix :
```

Ecran de réponse

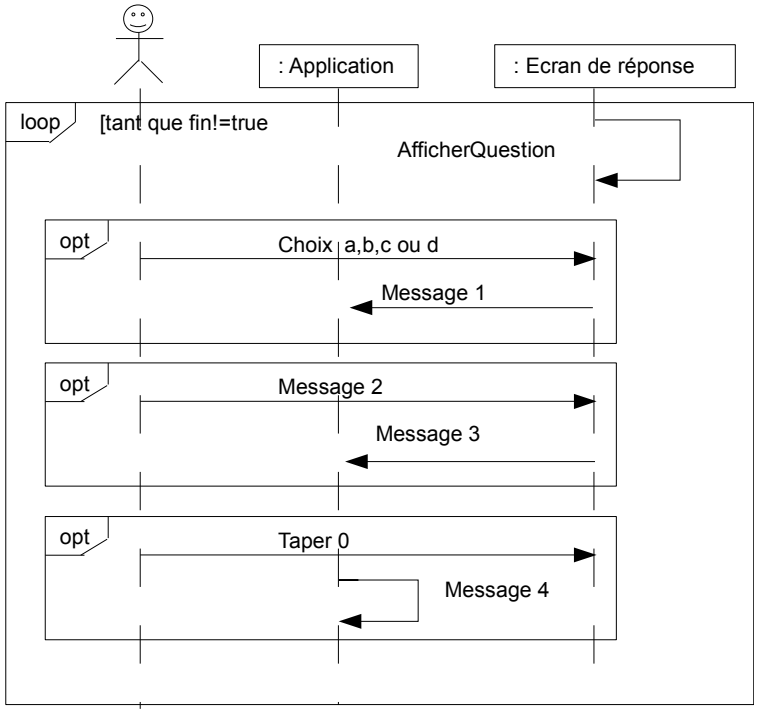
```
QCM_POO.txt

7) Soit le diagramme de classe suivant :
 --------          ------
| Machin |<>------| Truc |
 --------          ------
On peut affirmer que :

 a) Machin est un composant de Truc
*b) Truc sera un objet dynamique
*c) Il faut un destructeur dans Machin
 d) Truc contrôle Machin.

 a-d : (dé)sélectionner une réponse
   s : question suivante
   p : question précédente
   0 : revenir au menu
Entrer votre choix :
```

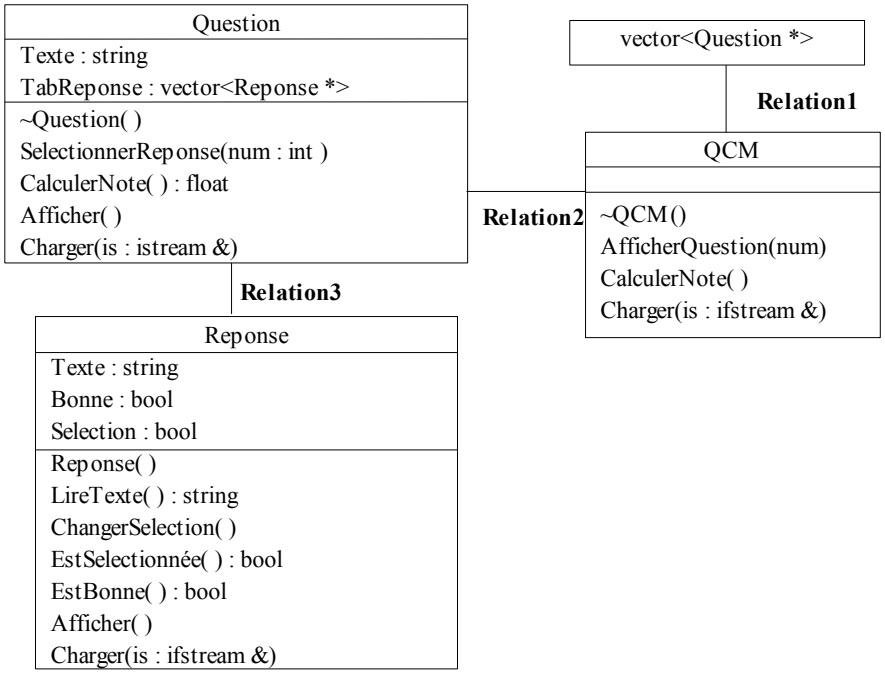
Sachant que les rectangles « loop » représentent des boucles de type « tant-que », et que les rectangles « opt » représentent des parties optionnelles, le diagramme de collaboration correspondant au scénario «répondre aux questions » est :



1) Le message *AfficherQuestion* :
   a) Est envoyé par l'objet application
   b) Affiche une question du QCM et demande le choix de l'utilisateur
   c) Est interne à l'objet Ecran de réponse
   d) Devrait aller de l'objet Ecran de réponse vers l'objet utilisateur
2) Le message Choix a,b, c ou d :
   a) Est envoyé par l'utilisateur
   b) Est une méthode de la classe Ecran de réponse
   c) Correspond à une série de réponses possibles de l'utilisateur
   d) Représente la saisie d'un caractère par l'utilisateur
3) A quel(s) message(s) peut correspondre *Message 1* ?
   a) Revenir au menu
   b) Changer de question
   c) (Dé)Sélectionner une réponse
   d) Charger le QCM
4) A quel(s) message(s) peut correspondre *Message 2* ?
   a) Charger le QCM
   b) Changer de question
   c) Afficher une question
   d) Taper s ou p
5) A quel(s) message(s) peut correspondre *Message 3* ?
   a) (Dé)sélectionner une réponse
   b) Changer de question
   c) Revenir au menu
   d) Taper s ou p
6) A quel(s) message(s) peut correspondre *Message 4* ?
   a) fin = true
   b) Enregistrer la réponse
   c) Passer à la question suivante
   d) Valider le QCM

Partie 2 : Conception

On donne un diagramme de classes de l'application avec les champs et les méthodes associées aux relations. Certaines autres méthodes ne figurent pas et certaines informations ont été volontairement omises.



7) De quel type peut être *Relation1* ?
   a) Une agrégation
   b) Une association
   c) Une composition
   d) Un héritage
8) Quel champ faut-il rajouter dans la classe QCM ?
   a) TabQuestion : vector <Question>
   b) TabQuestion : vector <string>
   c) TabQuestion : vector <Question *>
   d) aucun
9) De quel type peut être la relation 2 ?
   a) Une composition multiple
   b) Une agrégation multiple
   c) Une association multiple
   d) Un héritage
10) De quel type peut-être la relation 3 ?
   a) Une composition multiple
   b) Une agrégation multiple
   c) Une composition multiple
   d) Un héritage
11) A quoi peut servir le champ *Selection* ?
   a) Savoir si l'utilisateur à répondu

b) Savoir si la réponse est bonne
c) Savoir si la réponse a été sélectionnée
d) Compter le nombre de réponses sélectionnées

12) A quoi sert la classe *QCM* ?
a) Associer les réponses aux différentes questions
b) Regrouper les méthodes de gestion du QCM
c) Sélectionner une réponse
d) Charger un QCM depuis un fichier texte

13) Pourquoi avoir choisi un *vector* pour le champ *TabReponse* ?
a) Cela permet d'inserer ou de supprimer des réponses efficacement
b) Pour pouvoir accéder rapidement à n'importe quelle réponse
c) Pour gérer un nombre variable de réponses
d) Pour pouvoir utiliser un itérateur

14) Que fait la méthode *Charger* de la classe *Question* ?
a) Elle lit sur le flux le texte de la question
b) Elle compare la réponse de l'utilisateur à la bonne réponse
c) Elle charge la réponse de l'utilisateur
d) Elle lit sur le flux le texte des réponses

Partie 3 : Implantation

Ci dessous le code incomplet d'une méthode :

```
void Reponse::Charger(istream &is)
{
 string txt;
 à_completer_1
 is.get();
 if (txt.size()>0 && à_completer_2)
 {
   Bonne=true;
   à_completer_3
 }
 Texte=txt;
}
```

15) Le texte *à_compléter_1* peut être remplacé par :
a) is.read((char *)&txt,sizeof(string));
b) txt=is.get();
c) txt=getmultiline(is,'$');
d) getline(is,txt,'$');

16) Le texte *à_compléter_2* peut être remplacé par :
a) txt=="#"
b) txt!="#"
c) txt[0]=='#"
d) txt[0]!='#

17) Le texte *à_compléter_3* peut être remplacé par :
a) txt.pop_front();
b) txt[0]=txt[1];
c) txt=txt.substr(1,string::npos);
d) txt=txt.find_first_of('#')+1;

Ci dessous le code incomplet d'une méthode :

```
void Question::Charger(istream &is)
{
 getline(is,Texte,'$');
 is.get();
 bool fin=false;
 while(!fin)
 {
   Reponse *p= à_completer_1;
   p.Charger(is);
   if (is.eof()) fin=true;
   else à_completer_2
 }
}
```

18) A quoi sert l'instruction *is.get();* ?
a) Ignorer le prochain caractère du flux.
b) Terminer la saisie du texte.
c) À rien
d) Sauter le caractère fin de ligne.

19) Par quoi peut-on remplacer *à_completer_1* :
a) Reponse( )
b) new Reponse
c) NULL
d) GetReponse( )

20) Par quoi peut-on remplacer *à_completer_2 :*
a) TabReponse.resize(1,p);
b) TabReponse.push_back(p);
c) TabReponse.push_back(p.LireTexte());
d) TabReponse.push_back(*p);