Génie Logiciel Avancé

Année 2016-2017

Delphine Longuet, Thibaut Balabonski, Robin Pelle, Hadi Zaatiti
longuet@lri.fr, blsk@lri.fr, pelle@lri.fr, hadizaatiti@gmail.com

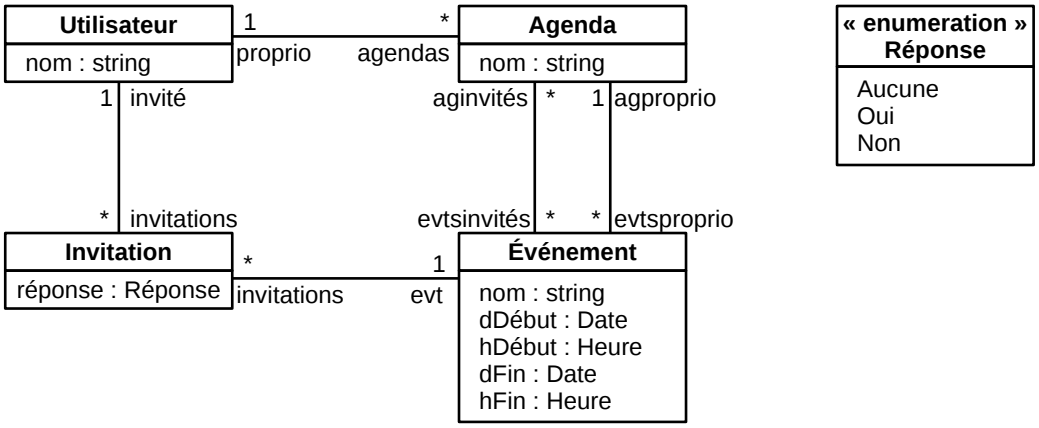
Partiel

26 octobre 2016

Seuls les transparents du cours (éventuellement annotés) sont autorisés.

Exercice 1 (barème indicatif : 12 points)

On considère le diagramme de classes suivant modélisant l'application web Memento.



Memento permet à ses utilisateurs de gérer des agendas, des événements et des invitations à des événements. Un utilisateur est propriétaire d'un certain nombre d'agendas et d'événements dans ces agendas. Un agenda a un unique propriétaire. Les agendas d'un utilisateur sont identifiés de façon unique par leur nom. Deux utilisateurs peuvent donc avoir des agendas de même nom, mais les noms des agendas d'un utilisateur donné sont tous différents. Un événement appartient à un seul agenda lorsqu'il est créé, son agenda propriétaire. Un événement a un nom, ainsi que des dates et heures de début et de fin définissant un créneau horaire valide. Pour simplifier, on considèrera que les utilisateurs sont identifiés de façon unique par leur nom.

Un utilisateur peut inviter d'autre utilisateurs de Memento à un événement dont il est propriétaire. Il ne peut pas s'inviter lui-même, il est déjà considéré comme participant à l'événement. Une invitation relie de façon unique un utilisateur invité à un événement et cet événement. La réponse par défaut à la création de l'invitation est `Aucune`. Ensuite, si l'invité répond qu'il participera à l'événement, celui-ci est ajouté dans un de ses agendas (au choix de l'invité). L'événement apparaît alors dans cet agenda en tant qu'événement invité. Si l'invité répond à l'invitation par la négative, l'événement n'est pas ajouté à ses agendas. Un invité peut revenir sur sa décision autant de fois qu'il le souhaite. Un utilisateur ne peut pas être invité plusieurs fois au même événement. Un utilisateur ne peut pas inviter d'autres utilisateurs à un événement où il est invité, quelle que soit sa réponse.

On suppose qu'on a les prédicats de comparaison habituels sur les dates et les heures, notés $=$, $<$, $\leq$, etc.

Invariants.

1. Parmi les invariants potentiels ci-dessous, déterminer :
   - ceux qui ne sont pas des invariants (et pourquoi) ;
   - ceux qui sont directement exprimés dans le diagramme de classes (préciser alors la combinaison d'attributs, d'associations, de cardinalités, etc. mise en jeu) ;
   - ceux qui ne sont pas exprimés dans le diagramme. Les formaliser en se servant de la navigation dans le diagramme de classes.

   (a) Les noms des utilisateurs sont uniques.

   (b) Un événement a un unique utilisateur propriétaire.

   (c) Tous les agendas ont des noms différents.

   (d) Tous les agendas d'un utilisateur ont des noms différents.

   (e) Dans un agenda, un événement ne peut pas apparaître à la fois en tant qu'événement dont l'utilisateur est propriétaire et événement auquel l'utilisateur est invité.

   (f) Les date et heure de fin d'un événement sont postérieures à ses date et heure de début.

   (g) Tous les événements auxquels un utilisateur est invité apparaissent dans ses agendas.

   (h) Si la réponse d'un utilisateur à une invitation est positive, alors l'événement correspondant apparaît dans un de ses agendas.

2. Pour chacune des phrases mathématiques suivantes, la formuler en français, puis dire si elle décrit un invariant du système ou non et pourquoi.

   (a) $\forall u \in \mathsf{Utilisateur}, u.\mathsf{agendas.evtsproprio} \cap u.\mathsf{invitations.evt} = \emptyset$

   (b) $\forall u \in \mathsf{Utilisateur}, \forall e \in u.\mathsf{agendas.evtsproprio}, \exists i \in u.\mathsf{invitations}, i.\mathsf{réponse} = \mathsf{Oui} \wedge i.\mathsf{evt} = e$

   (c) $\forall u \in \mathsf{Utilisateur}, |u.\mathsf{invitations}| = |u.\mathsf{invitations.evt}|$

Spécification des opérations.

3. Donner une spécification formelle en termes de pré et post-conditions pour les opérations suivantes associées au diagramme de classes.

   (a) Utilisateur :: accepter(i : Invitation, a : Agenda) change la réponse de l'utilisateur à l'invitation par Oui et ajoute l'événement correspondant à l'agenda. Il faut que l'invitation concerne l'utilisateur et n'ait pas déjà été acceptée, et que l'agenda appartienne à l'utilisateur. La réponse de l'invitation est mise à jour et l'événement est ajouté à l'agenda en tant qu'événement invité.

   (b) Utilisateur :: inviter(u : Utilisateur, e : Événement) invite l'utilisateur passé en argument à l'événement. Il faut que l'utilisateur sur lequel l'opération est appelée (*this*) soit propriétaire de l'événement, qu'il soit différent de l'utilisateur u et que u ne soit pas déjà invité à cet événement. Après l'opération, il existe une invitation reliant l'utilisateur u à l'événement e, dont la réponse est Aucune, et qui n'existait pas dans l'état précédent.

4. On donne la spécification suivante d'une opération. Formuler cette spécification en français et expliquer l'opération décrite (5 lignes maximum).
   Événement :: operation() : $\mathcal{P}$(Utilisateur)
   pre : *true*
   post : $result = \{u \in this.\text{invitations.invité} \mid \forall i \in this.\text{invitations} \cap u.\text{invitations}, i.\text{réponse} = \text{Oui}\} \cup this.\text{agproprio.proprio}$

Exercice 2 (barème indicatif : 8 points)
On considère une opération qui prend un tableau de caractères s en argument et renvoie vrai si et seulement si un caractère a apparaît strictement plus de fois qu'un caractère b dans s.

```
boolean comp_occurrences(char s[], char a, char b)
```

1. On propose les données de test suivantes.

	s	a	b
1	"terrestre"	'r'	't'
2	"minimum"	'f'	'u'
3	"pierre"	'e'	'e'
4	"aïeux"	'g'	'l'

   Pour chacun de ces tests, donner l'objectif visé par le test ainsi que le résultat attendu.
2. Compléter le jeu de tests précédent de manière à couvrir le mieux possible les différents cas de la spécification. Pour chaque nouveau test, donner son objectif, des données d'entrée concrètes et le résultat attendu.
3. Dans les tests des questions 1 et 2, identifier les tests aux limites et expliquer pour chacun d'eux la limite visée. Ajouter ceux qui manqueraient.
4. On considère l'implantation suivante de l'opération.

```
boolean comp_occurrences(char s[], char a, char b) {
    int res = 0;
    for(int i = 0; i < s.length; i++) {
        if(s[i] == a) {
            res ++;
        } else if(s[i] == b) {
            res--;
        }
    }
    return (res > 0);
}
```

   Cette implantation réussit-elle tous les tests des questions 1 et 2 ? Donner le ou les tests sur lesquels elle échoue et expliquer la ou les fautes découvertes par ces tests. Quelle modification faut-il apporter au programme pour corriger la faute ?