LO43-2013A - FS01-01

FINAL LO43 – Automne 2013

Durée 2 heures – Documents interdits Ordinateurs portables interdits -

Les parties sont à rendre sur des copies séparées

I. PARTIE JAVA (10 POINTS):

Une application informatique a pour objectif de simuler le déplacement de robots sur un graphe. Le graphe est défini par un couple (*X*, *U*), avec *X* l'ensemble des sommets du graphe et *U* l'ensemble des arcs de ce graphe. Un arc de *U* est un couple (*a*, *b*) avec *a* et *b* deux sommets de *X*.

La figure 1 représente le graphe G=(X,U) avec les sommets X={s1, s2, s3, s4, s5} et les arcs U={(s1,s2), (s1, s3), (s2, s5), (s3, s4), (s4, s2), (s5, s4)}.

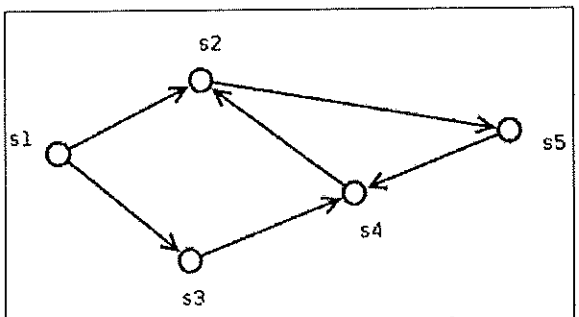


Figure 1: Exemple de graphe

1. Les contraintes de l'application font que le stockage des sommets et des arcs du graphe (classe **CNode** et **CEdge** du code 1) doit se faire dans une liste chaînée héritant de la classe **CLinkedList** (code 2) sans utiliser les classes de « java.util ». On propose d'écrire 2 listes chaînées pour stocker, d'une part, les sommets du graphe (1ère liste) et d'autre part les arcs (2ème liste).

Définir la classe **CLinkedListNodes** (resp. **CLinkedListEdges**) qui permet de stocker les sommets (resp. les arcs) du graphe en précisant les méthodes « **add**(Object *obj*) » qui permet d'ajouter un élément « *obj* » à la liste et « **get**(int *i*) » qui permet de récupérer l'objet d'indice *i* (on supposera que le premier objet stocké est d'indice 1, le deuxième d'indice 2, etc...).

2. En n'utilisant que des types simples ou des classes existantes de l'application, modifier la classe « **CNode** » afin de représenter que zéro, un ou plusieurs arcs peuvent partir de ce sommet. Définir les deux méthodes :
   - « int **getNbrEdges**() » : qui renvoie le nombre d'arcs sortants du sommet
   - « CEdge **getEdge**(int i) » : qui renvoie l'arc d'indice *i* sortant du sommet, avec *i* compris entre 1 et **getNbrEdges**().
3. Chaque robot se déplaçant sur le graphe sera un thread java qui partira d'un sommet *s* du graphe, suivra aléatoirement un arc *a* partant de ce sommet *s*, pour arriver au sommet d'arrivée de l'arc **a** et ainsi de suite. Les contraintes dimensionnelles des robots font qu'un seul robot au plus peut se trouver sur un sommet du graphe à un instant donné, c'est à dire qu'il n'est pas possible à deux robots ou plus de se trouver sur le même sommet à un instant donné. On supposera que :

- Le temps de parcours d'un robot sur un arc est une variable aléatoire comprise entre « **edgeTimeMin** » et « **edgeTimeMax** », qui représentent le temps minimum et maximum de

parcours des arcs en millisecondes (ces constantes sont communes à tous les robots).

- De la même manière, le temps d'attente d'un robot à un sommet sera une variable aléatoire comprise entre « **nodeTimeMin** » et « **nodeTimeMax** », qui représentent le temps minimum et maximum d'attente aux sommets en millisecondes (ces constantes sont communes à tous les robots).

On pourra utiliser la méthode «Math.random()» pour tirer un nombre aléatoire entre 0 et 1.

Ecrire la classe « **Robot** » en précisant ses attributs ainsi que toutes les méthodes nécessaires à son fonctionnement.

4. On veut tester la possibilité de fusionner les deux listes chaînées **CLinkedListNodes** et **ClinkedListEdges** en une seule liste chaînée, qui s'appellera **CLinkedListGraph**, qui contiendra tous les sommets et les arcs du graphe.

   a) Quel problème se pose alors lorsque l'on veut récupérer un sommet ou un arc de cette liste ?

   b) Proposer une solution pour résoudre ce problème.

```
/** Sommet d'un graphe */
class CNode {
        private int node ; /* numéro du sommet */

        public CNode (int n) {node=n ;}
        public int getNode() { return node; }
}

/** Arc d'un graphe */
class CEdge {
        private CNode start ;
        private CNode end ;

        public CEdge (CNode s, CNode e) {start=s;end=e;}

        public CNode getStart() { return start; }
        public CNode getEnd() { return end; }
}
```

Code 1. Classes CNode et CEdge

```
/** Liste chaînée générique */
abstract public class CLinkedList {
        protected Object obj ;
        protected CLinkedList next ;

        public CLinkedList() { obj=null; next=null; }
        abstract public void add(Object o) ;
        abstract public Object get(int i) ;

}
```

*Code 2. Classe **CLinkedList***

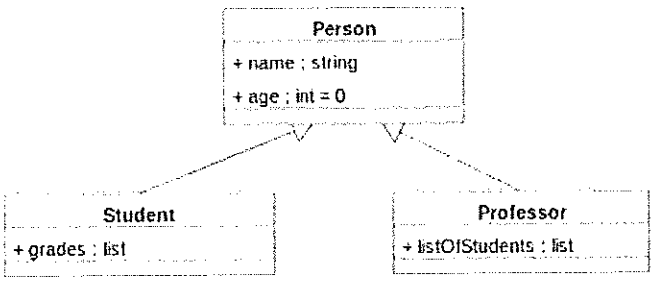
II. Partie UML (10 points):

Exercice (8 Points): Mise en Abîme

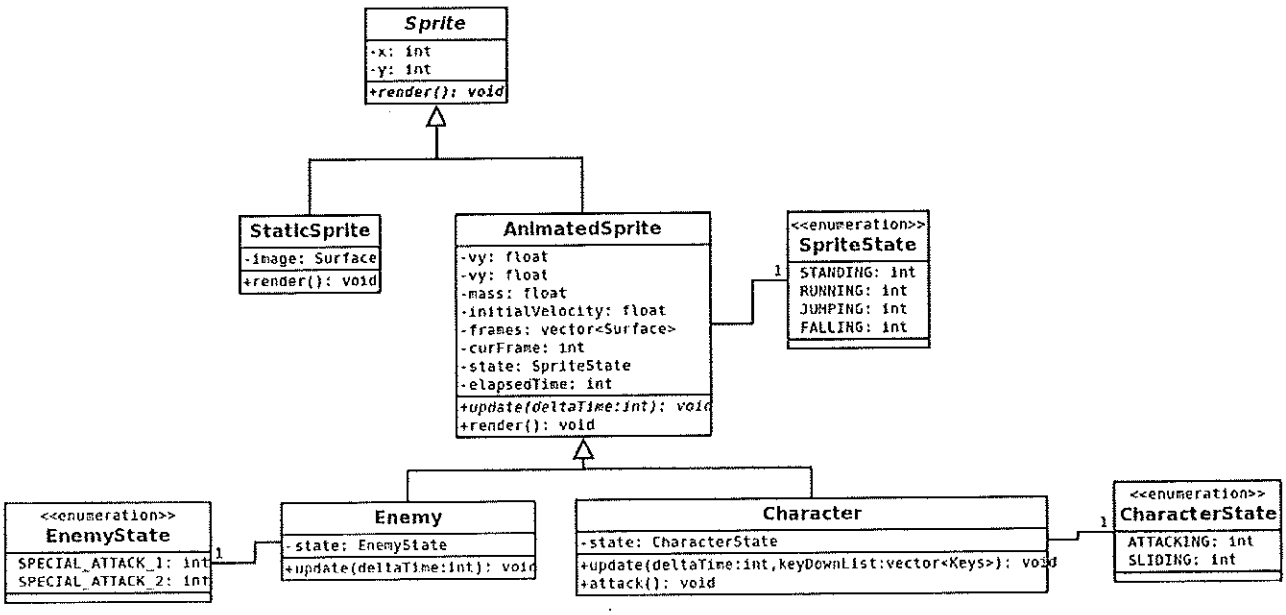
On veut faire un logiciel permettant de dessiner uniquement des diagrammes de classes UML. Pour cet exercice, nous nous intéresserons qu'à la partie modèle. (On ne s'occupera pas de la partie interface homme/machine).

Question 1: Ecrire le diagramme de classe de ce logiciel. (3 pts)

Question 2: Ecrire le diagramme d'objets, respectant votre proposition de la question 1, du diagramme de classe suivant (2 pts):



Question 3: Ecrire le diagramme d'objets du diagramme de classe suivant (3 pts):



Question de cours (2 points): Quels sont les diagrammes UML permettant d'expliquer l'aspect dynamique d'un programme.