Examen Final de LO43 : Programmation Orientée Objet

Semestre A2014 Durée : 2h

Documents, calculatrices, téléphones portables interdits. Dictionnaires papiers autorisés.

Questions de Cours (4 pts)

Question 1

Quelles sont les différences entre l'ellipse en Java et celle du C++ ? Sur quel concept repose l'ellipse en Java ?

Question 2

Comment la notion d'amitié est faite en Java ? Le mécanisme est-il plus précis que celui du C++ ? Justifiez.

Exercices Java (4 pts)

Exercice 1

Qu'affiche le programme suivant ?

```
class A {
        private int val=0;

public static void affichePlus (int a) {
         a++;
         System.out.println (a);
}

public static void affichePlus (A a) {
         a.val++;
         System.out.println (a.val);
}

public static void main (String[] args) {
        A uneClasse = new A ();
        A uneAutreClasse = new A();
        affichePlus (uneClasse.val);
        affichePlus (uneClasse.val);
        affichePlus (uneClasse);
        affichePlus (uneClasse);
        affichePlus (uneAutreClasse);
        affichePlus (uneAutreClasse);
        if (uneClasse==uneAutreClasse) System.out.println("Egales"); else
System.out.println("Differentes");

} }
```

Exercice 2

Les exemples suivants sont-ils corrects ? Justifiez.

```
1. class A {
   private int a;
```

```
            public A() { System.out.println(a); }
            public A(int a) { this.a = a; this(); }
    }
```

2.
```
class A {
int a;
}
class B extends A {
int b;
B(int a, int b) { this.a = a; this.b = b; }      }
```

3.
```
class A {
int a;
A(int a) { this.a = a; }
}

class B extends A {
int b;
B(int a,int b) { this.a = a; this.b = b; }
}
```

4.
```
class A {
private int a;
A() {this.a=0;}
}
class B extends A {
private int b;
B() {this.a=0; this.b=0;}
}
```

Robots' Hunger Games (12 pts)

On veut faire un jeu de combat de robots dans une arène. Les robots peuvent être de plusieurs types (avec des pattes, avec des roues, avec des chenilles). Les robots ont cependant des comportements de base communs tels que : avancer, reculer, tourner, attaquer et défendre. Lorsqu'un robot attaque un adversaire, la réussite de cette action dépend d'une probabilité P. Cette probabilité a une valeur initiale qui dépend du type de robot. Cette valeur peut changer en fonction du temps de jeu du robot et de l'état de l'adversaire attaqué. Ainsi, à chaque pas de temps P diminue de 5%. De même P est diminuée de 50% si l'adversaire se défend. L'objectif d'un robot est de survivre le plus longtemps possible dans l'arène. Chaque robot possède une intelligence artificielle qui fait appel aux comportements de base. Le joueur peut influer sur le jeu en donnant des conseils à l'intelligence artificielle d'un robot, en fournissant des objets aux robots permettant de faire varier la probabilité P de chacun ou en introduisant des catastrophes affectant tous les robots. L'objectif de cet exercice est de concevoir ce jeu de combat respectant ces règles.

Partie 1 : Concevoir le jeu

Question 1

Donner le diagramme de cas d'utilisation de ce jeu.

Question 2

On souhaite que tous les robots possèdent les mêmes comportements de base (utilisant les mêmes prototypes de méthodes). Comment imposer ceci d'un point de vue programmation orientée objet ?

Question 3

On veut synchroniser le fonctionnement des robots. C'est à dire que, à chaque pas de temps, on veut exécuter la même méthode pour chacun des robots. Comment faire ceci proprement en programmation? Quel mécanisme de programmation orientée objet permet cette réalisation ? Faut-il avoir recours obligatoirement aux Threads sachant que l'on peut avoir jusqu'à une dizaine de robots.

Question 4

Ecrire le diagramme de classe de ce jeu. Vous prendrez soin de détailler les prototypes des méthodes.

Question 5

L'élément clé de ce jeu est la résolution des combats. Ecrire le diagramme de séquence permettant la réalisation de ces combats.

Partie 2 : Les robots à pattes.

Cette partie de l'exercice se concentre sur le fonctionnement particulier des robots à pattes. En effet, l'un des éléments critique de ce type de robot est de maintenir une stabilité lors des déplacements. Pour cela, une règle essentielle doit être obligatoirement appliquée : **deux pattes successives ne doivent pas être en l'air simultanément**. Cette règle est valable pour les robots à 2, 4 et 6 pattes. Chaque patte possède deux mouvements possibles : la protraction (la patte est levée et se déplace vers l'avant du robot) et la retraction (la patte est au sol et se dirige vers l'arrière du robot). La classe Patte est fournie et commune à tous les robots à pattes. Elle possède les méthodes suivantes :

- `public Patte()` qui est le contructeur,
- `public void retracter (int percent)` qui permet à la patte de se retracter avec comme paramètre le pourcentage par rapport à la longueur maximale de retraction,
- `public void protracter()` qui effectue la phase de protraction.

Chaque objet Patte est contrôlé par un objet de la classe `ControlleurPatte` qui appelle les méthodes de Patte au bon moment. La synchronisation des objets de type `ControlleurPatte` se fait par accès à un objet commun, instance de la classe `Privilege`, permettant de gérer la contrainte du nombre de pattes levées. L'utilisation de cet objet est analogue à la ressource partagée dans les problèmes de type Producteur/Consommateur.

Question 6

Modifier le diagramme de classe de la question 4 pour prendre en compte ces nouvelles contraintes.

Question 7

Ecrivez le diagramme d'objets correspondant à un robot à quatre pattes

Question 8

Ecrivez le code Java des classes `ControleurPatte` et `Privilege`.