**TP2 : les comportements de l’agent**

**I. Introduction**

Dans la programmation orientée agent, les différentes tâches d'un agent doivent obéir à certaines règles et doivent être écrite sous une forme compréhensible par la plateforme utilisée. Nous allons voir dans ce qui suit comment définir ces tâches et les utiliser dans JADE

**II. Vue générale**

Pour qu'un agent JADE exécute une tâche, nous avons tout d'abord besoin de définir ces tâches. Les tâches dans JADE (appelées des **behaviours** ou des **comportements**) sont des instances de la classe **jade.core.behaviours** . pour qu'un agent exécute une tâche on doit lui l'attribuer par la méthode **addBehaviour(Behaviour b)** de la classe **jade.core.Agent**.   
Chaque Behaviour doit implémenter au moins les deux méthodes :

* **action() :** qui désigne les opérations à exécuter par le Behaviour;
* **done() :** qui exprime si le Behaviour a terminé son exécution ou pas.

Il existe deux autres méthodes dont l'implémentation n'est pas obligatoire mais qui peuvent être très utiles :

* **onStart() :** appelée juste avant l'exécution de ma méthode action();
* **onEnd() :** appelée juste après la retournement de **true** par la méthode done().

Des fois on a besoin de savoir quel est le propriétaire d'un Behaviour, et cela peut être connu par le membre **myAgent** du Behaviour en question.   
JADE alloue un thread par agent, pour cela un agent exécute un Behaviour à la fois. L'agent peut exécuter plusieurs Behaviours simultanément en choisissant un bon mécanisme de passation d'un Behaviour à un autre (c'est à la charge du programmeur et non pas à la charge du JADE).

**III. Les Behaviours du JADE**

**III-A. Les Behaviours simples**

JADE offre trois types de Behaviours simple. Ces Behaviours sont :

**III-A-1. One-shot Behaviour**

Un one-shotBehaviour est une instance de la classe **jade.core.behaviours.OneShotBehaviour**. Il a la particularité d'exécuter sa tâche une et une seule fois puis il se termine. La classe OneShotBehaviour implémente la méthode **done()** et elle retourne toujours **true**.

**III-A-2. CyclicBehaviour**

Un cyclicBehaviour est une instance de la classe **jade.core.behaviours.CyclicBehaviour**. Comme son nom l'indique un cyclicBehaviour exécute sa tâche d'une manière répétitive. La classe CyclicBehaviour implémente la méthode **done()** qui retourne toujours **false**.

**III-A-3. GenericBehaviour**

Un GenericBehaviour est une instance de la classe **jade.core.behaviours.Behaviour**.LeGenericBehaviour vient entre le One-shotBehaviour et le CyclicBehaviour de faite qu'il n'implémente pas la méthode done() et laisse son implémentation au programmeur, donc il peut planifier la terminaison de son Behaviour selon ces besoin.

**III-A-4. Exemple**

Voici un exemple sur l'utilisation des Behaviours simples

importjade.core.Agent;

importjade.core.behaviours.Behaviour;

importjade.core.behaviours.CyclicBehaviour;

importjade.core.behaviours.OneShotBehaviour;

public class SimpleAgent extends Agent {

protected void setup() {

// l'ajout d'un one-shot behaviour pour afficher un Hello world :D

addBehaviour(new OneShotBehaviour(this){

public void action(){

System.out.println("Bonjour je suis l'agent "+getLocalName());

}

});

// l'ajout d'un CyclicBehaviour pour afficher un message à chaque fois

//qu'il s'execute

addBehaviour(new CyclicBehaviour(this) {

public void action() {

System.out.println("cyclique... ");

}

});

// l'ajout d'un genericbehaviour

// leBehaviour s'arrête quand aléatoire reçoit la valeur 7

addBehaviour(new RandomBehaviour());

}

/\*\*

\* Inner class RandomBehaviour

\*/

private class RandomBehaviour extends Behaviour {

privateintaleatoire ;

public voidaction() {<

aleatoire = (int) (Math.random()\*10);

System.out.println("aleatoire ="+ aleatoire);

}

publicboolean done() {

returnaleatoire == 7;

}

publicintonEnd() {

myAgent.doDelete();

returnsuper.onEnd();

}

}

}