**TP3 Les comportement de l’agent (suite)**

**1 . Les Behaviours planifiés**

Pour planifier une tâche d'un agent JADE offre deux types de Behaviours :

**1.1. WakerBehaviour**

Le WakerBehaviour est implémenté de façon à exécuter la méthode **onWake()** après une période passée comme argument au constructeur. Cette période est exprimée en millisecondes. Le Behaviour prend fin juste après avoir exécuté la méthode **onWake()**.

**1. 2. TickerBehaviour**

Le TickerBehaviour est implémenté pour qu'il exécute sa tâche périodiquement par la méthode **onTick()**. La durée de la période est passée comme argument au constructeur.

**Exemple**

Voici un exemple d'un compte à rebours:

1. Le premier Behaviour affiche le temps restant chaque seconde.

2. Le deuxième arrête le compte à rebours et termine l'agent.

Import jade.core.Agent;

importjade.core.behaviours.WakerBehaviour;

importjade.core.behaviours.TickerBehaviour;

public class CompteaRebours extends Agent {

protected void setup() {

final int nombreDeSecondes = (int)(Math.random()\*15);

System.out.println("compteur a rebours de "+nombreDeSecondes);

//ce Behaviour montre le temps restant

addBehaviour(new TickerBehaviour(this, 1000) {

protected void onTick() {

System.out.println("Il reste "+(nombreDeSecondes-getTickCount())+" seconds"); } });

 //ce Behaviour va arreter le compte et terminer l'agent

addBehaviour(new WakerBehaviour(this, 1000\*nombreDeSecondes) {

protected void handleElapsedTimeout() {

System.out.println("Terminé");

myAgent.doDelete();

 } });} }

**Exercice I**

Combinez le ticker behaviour et le oneShot behaviour pour avoir un cyclique behaviour, et combinez le waker dans le cyclique pour le rendre oneShot behaviour.

**Les comportements de l’agent (suite)**

**2 Les Behaviours composés**

JADE offre un ensemble de Behaviours composés qui servent à présenter des tâches complexes. La classe mère de toutes les autres complexes est la classe **jade.core.behaviours.CompositeBehaviour**.
Une instance de cette classe, un Behaviour bien entendu, contient **des sous-Behaviours**, la méthode **action()** est déjà implémentée et invoque à chaque fois la méthode **action()** de l'un de ses sous-Behaviour. L'ordre d'exécution des Behaviours est à la charge des deux méthodes **scheduleFirst()** et **scheduleNext()** que les classes héritant de la classe **CompositeBehaviour** doivent implémenter.

Il existe trois classes filles:

**1. SequentielBehaviour**

**2. FSMBehaviour**

**3. ParallelBehaviour**

**1. SequentielBehaviour**

Le Behaviour commence par exécuter le premier sous-Behaviour et lorsque celui-là termine son exécution (sa méthode **done()** retourne true), il passe au prochain Behaviour, et ainsi de suite.

Les sous-Behaviours sont ajoutés au **sequentielBehaviour** par la méthode **addSubBehaviour()**. L'ordre de l'ajout détermine l'ordre d'exécution.
Voici un exemple :

importjade.core.Agent;

importjade.core.behaviours.OneShotBehaviour;

importjade.core.behaviours.SequentialBehaviour;

importjade.core.behaviours.WakerBehaviour;

public class Seq extends Agent {

protected void setup(){

 SequentialBehaviourcomportementSequentiel = new SequentialBehaviour();

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public void action() {

 System.out.println("le premier sous-comportement");

 }

 });

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public voidaction() {

 System.out.println("le second sous-comportement");

 }

 });

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public voidaction() {

 System.out.println("le derniers sous-comportement");

 myAgent.doDelete();}

 });

 addBehaviour(comportementSequentiel);

 }}

**Exercice II**

4 Behaviors qu'on veut exécuter séquentiellement, chacun affiche un message pendant x secondes (10, 20, 5 et 15). Affiche à la fin le nombre total des secondes.