Université Abdelhamid Mehri Constantine 2- Algérie
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département d'Informatique Fondamentale et ses Applications

1ère Année Master Sciences et Technologies de l'information et de la Communication (M1STIC)

TP Algorithmes distribués (ALDI)

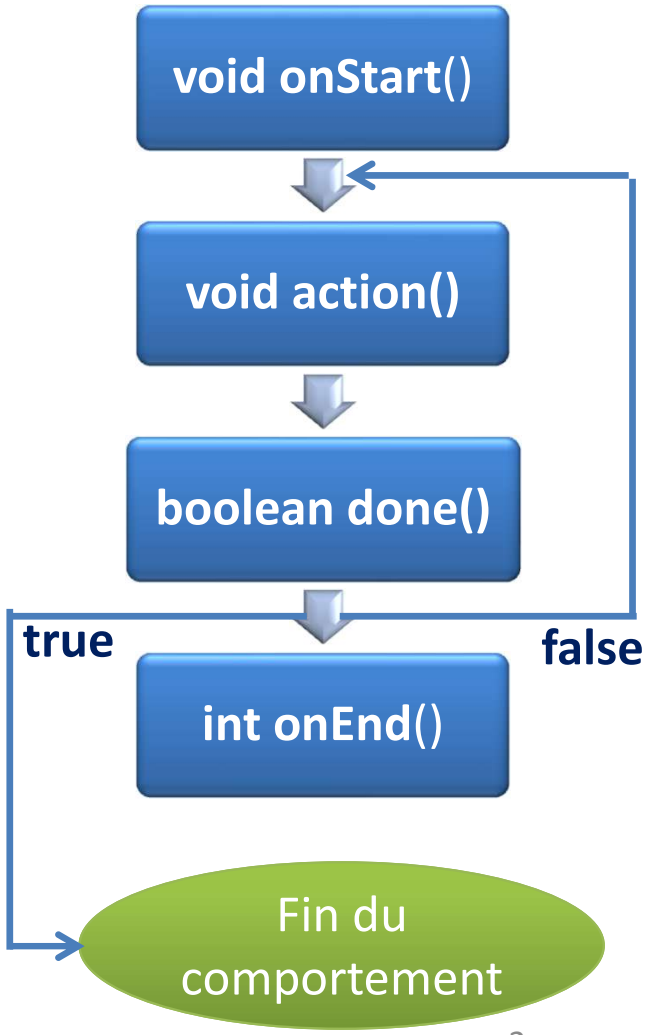
TP 02 : Initiation à la plateforme JADE (Les comportements)

Année universitaire : 2023/2024

Les comportements

- Dans la plateforme **JADE**, un agent possède un ou plusieurs comportements (Behaviours) qui définissent ses actions
- Un comportement hérite de la classe **jade.core.behaviours**
- Chaque comportement doit implémenter au moins les deux méthodes :
  - ✓ void **action**() : désigne les opérations à exécuter par le comportement ;
  - ✓ boolean **done**() : indique si le comportement a terminé son exécution.
    - ➢ si la méthode **done()** retourne **false** alors le comportement n'a pas terminé son exécution
    - ➢ si la méthode **done()** retourne **true** alors le comportement a terminé son exécution

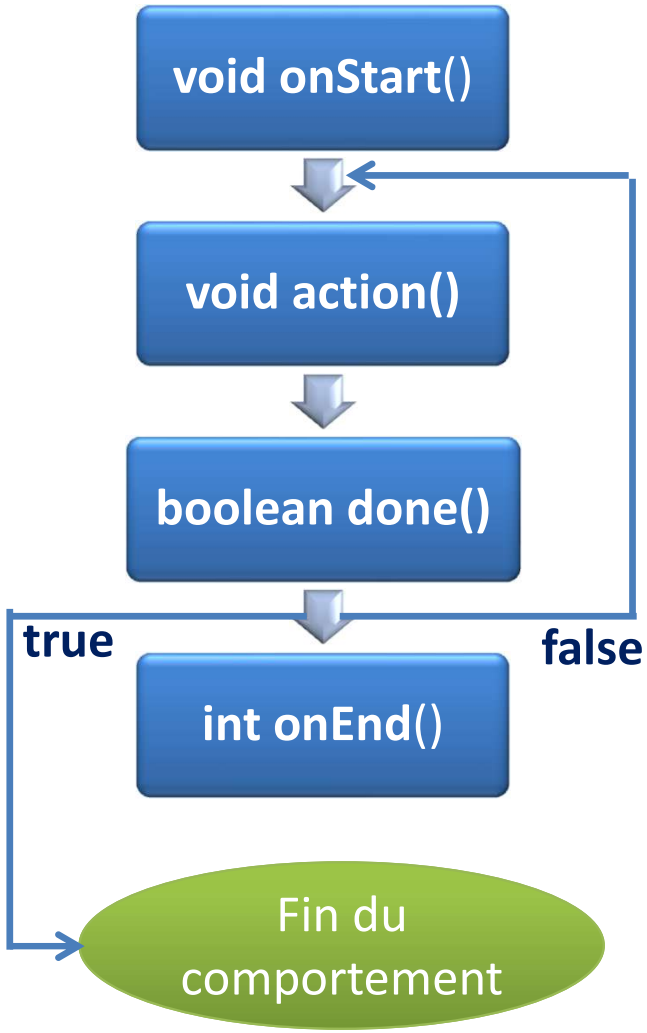
Cycle de vie d'un comportement



Les comportements

- Il existe d'autres méthodes dont l'implémentation n'est pas obligatoire mais qui peuvent être très utiles :
  - ✓void **onStart**() : appelée juste avant l'exécution de la méthode **action**();
  - ✓int **onEnd**() : appelée juste après le retournement de **true** par la méthode **done**().
- **L'ajout** d'un comportement à l'agent se fait par la méthode **addBehaviour**() .

Cycle de vie d'un comportement

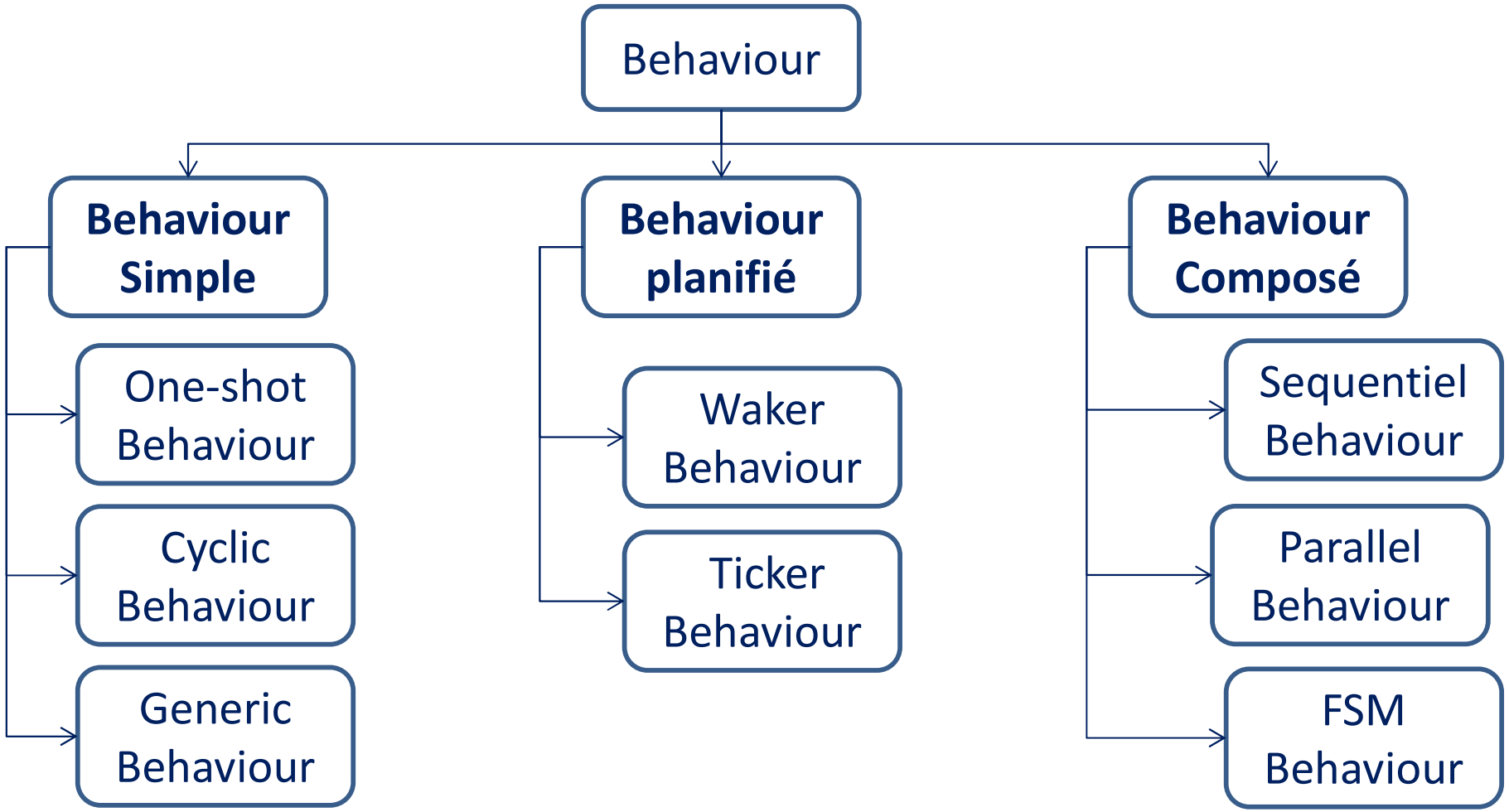


Les comportements

Il existe trois types de comportements dans la plateforme JADE :

- Comportement Simple
- Comportement Planifié
- Comportement Composé

Chacun de ces comportements est composé à son tour de plusieurs comportements

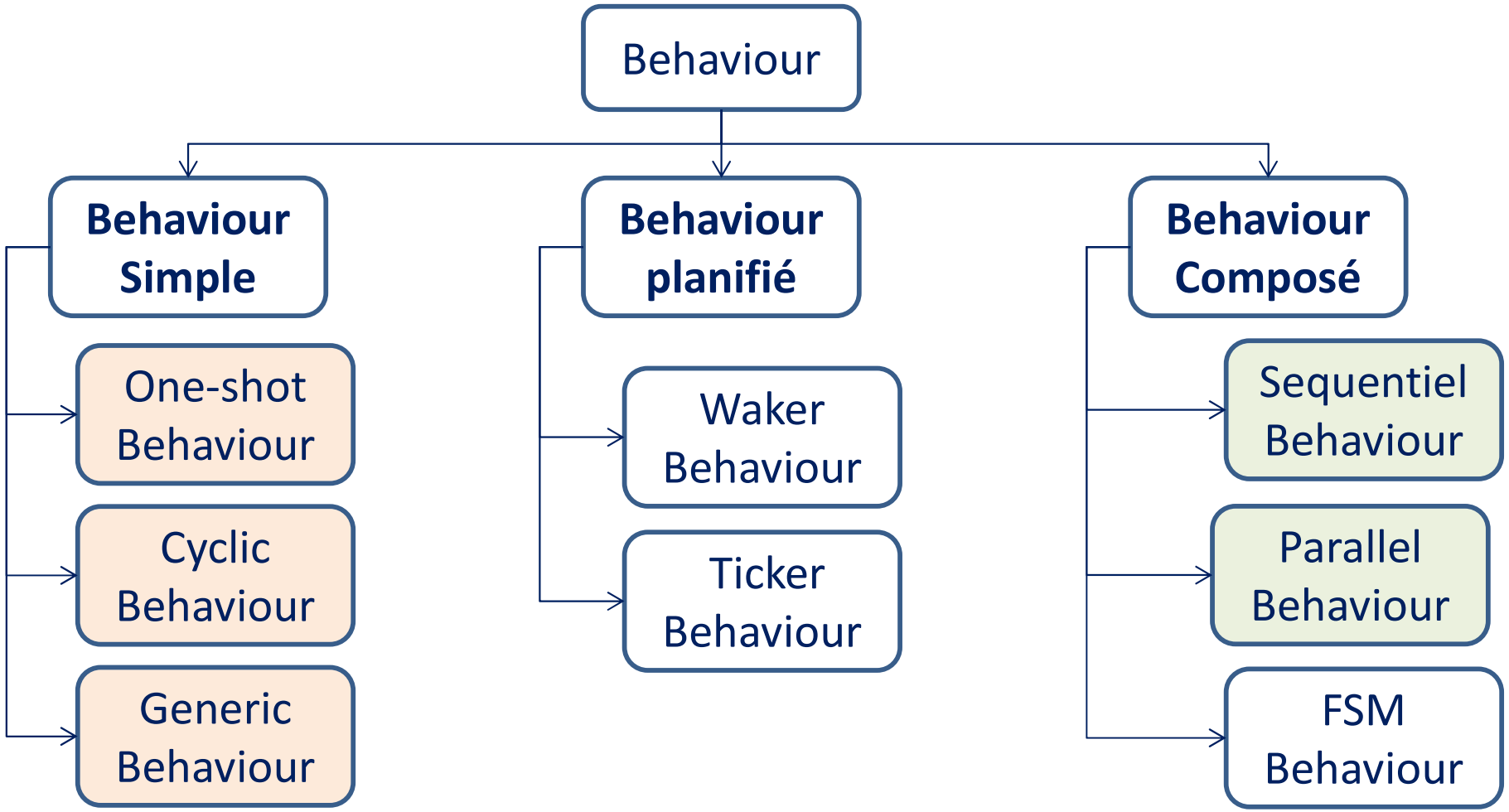


Les comportements

Il existe trois types de comportements dans la plateforme JADE :

- Comportement Simple
- Comportement Planifié
- Comportement Composé

Chacun de ces comportements est composé à son tour de plusieurs comportements



Les comportements

Les comportements Simples

- **OneShotBehaviour**
  - ✓est une instance de la classe : jade.core.behaviours.OneShotBehaviour
  - ✓exécute le comportement **une seule fois** puis il se termine.
  - ✓implémente la méthode **done**() et elle retourne toujours **true**.

Exemple:

```
public class Calculateur extends Agent {
   protected void setup() {
      System.out.println("Je suis l'agent : "+getLocalName());
      addBehaviour(new Addition());
   } //setup
   public class Addition extends OneShotBehaviour{
      public void action(){
         int a = (int)(Math.random() * 100);
         int b = (int)(Math.random() * 100);
         int c=a + b;
         System.out.println("Agent "+getLocalName()+" : j'ai calculé : "+a+"+"+b+"=" +c);
      } //action
   } // class Addition
} // class Calculateur
```

Les comportements

Les comportements Simples

- **OneShotBehaviour**

  Pour tester le programme :

  1. Créer un projet TPALDI02,
  2. Créer un package OneShotBehaviour,
  3. Dans le package OneShotBehaviour, créer une classe Calculateur et taper son code
  4. Dans le package OneShotBehaviour, créer une classe Test et taper le code suivant :

```
public class Test {
   public static void main(String[] args) {
      String [] commande = new String[3];
      String argument ="";
      argument = argument+"c1:OneShotBehaviour.Calculateur";
      //argument = argument+";c2:OneShotBehaviour.Calculateur";
      commande [0]="-cp";
      commande [1]="jade.boot";
      commande [2]= argument;
      jade.Boot.main(commande);
   }
}
```

Les comportements

Les comportements Simples

- **CyclicBehaviour**
  - ✓est une instance de la classe : jade.core.behaviours.CyclicBehaviour.
  - ✓exécute le comportement d'une manière **répétitive**.
  - ✓implémente la méthode **done**() qui retourne toujours **false**.

Exemple:

```
public class Calculateur extends Agent {
   protected void setup() {
      System.out.println("Je suis l'agent : "+getLocalName());
      addBehaviour(new Addition());
   } //setup
   public class Addition extends CyclicBehaviour {
      public void action(){
         int a = (int)(Math.random() * 100);
         int b = (int)(Math.random() * 100);
         int c=a + b;
         System.out.println("Agent "+getLocalName()+" : j'ai calculé : "+a+"+"+b+"=" +c);
      } //action
   } // class Addition
} // class Calculateur
```

Les comportements

Les comportements Simples

- **GenericBehaviour**
  - ✓est une instance de la classe : jade.core.behaviours.Behaviour.
  - ✓n'implémente pas la méthode **done**()
  - ✓L'implémentation doit être faite par le programmeur

Exemple:

```
public class Calculateur extends Agent {
   protected void setup() {
      System.out.println("Je suis l'agent : "+getLocalName());
      addBehaviour(new Addition());
   } //setup
   public class Addition extends Behaviour {
      int c;
      public void action(){
         int a = (int)(Math.random() * 100);
         int b = (int)(Math.random() * 100);
         c=a + b;
         System.out.println("Agent "+getLocalName()+" : j'ai calculé : "+a+"+"+b+"=" +c);
      } //action
```

Les comportements

Les comportements Simples

- **GenericBehaviour**
  - ✓est une instance de la classe : jade.core.behaviours.Behaviour.
  - ✓n'implémente pas la méthode **done**()
  - ✓L'implémentation doit être faite par le programmeur

Exemple: (suite)

```
    public boolean done(){
      return c == 100;
    } //done
  } // class Addition
} // class Calculateur
```

Les comportements

Les comportements Planifiés

- **WakerBehavior**
  - ✓exécute la méthode **onWake()** après une période passée comme argument au constructeur
  - ✓Cette période est exprimée en millisecondes.
  - ✓Le comportement prend fin juste après avoir exécuté la méthode **onWake()**.

Exemple:

```
public class Calculateur extends Agent {
   public void setup() {
      System.out.println("Agent : "+getLocalName());
      addBehaviour(new Addition(this,5000));
   }//setup
   private class Addition extends WakerBehaviour{
      int a, b, c;
      public Addition(Agent a, int durée){
         super(a, durée);
      }// Constructeur Addition
```

Les comportements

Les comportements Planifiés

- **WakerBehavior**
  - ✓exécute la méthode **onWake()** après une période passée comme argument au constructeur
  - ✓Cette période est exprimée en millisecondes.
  - ✓Le comportement prend fin juste après avoir exécuté la méthode **onWake()**.

Exemple: (suite)

```
        protected void onWake () {
            a = (int)(Math.random() * 100);
            b = (int)(Math.random() * 100);
            c = a + b;
            System.out.println("Agent"+getLocalName()+": j'ai calculé:"+a+"+"+b+"="+c);
        } // onWake
    } // class Addition
} // class Calculateur
```

Exécuter le programme en changeant la valeur de la durée d'attente

Les comportements

Les comportements Planifiés

- **TickerBehaviour**
  - ✓exécute sa tâche périodiquement par la méthode **onTick**().
  - ✓La durée de la période est passée comme argument au constructeur.

Exemple:

```
public class Calculateur extends Agent {
   protected void setup() {
      System.out.println("Je suis l'agent : "+getLocalName());
      addBehaviour(new Addition(this,5000));
   } //setup
   private class Addition extends TickerBehaviour{
      public Addition(Agent a, int durée){
         super(a, durée);
      }// Constructeur Addition
```

Les comportements

Les comportements Planifiés

- **TickerBehaviour**
  - ✓exécute sa tâche périodiquement par la méthode **onTick**().
  - ✓La durée de la période est passée comme argument au constructeur.

Exemple:

```
public class Calculateur extends Agent {
   protected void setup() {
      System.out.println("Je suis l'agent : "+getLocalName());
      addBehaviour(new Addition(this,5000));
   } //setup
   private class Addition extends TickerBehaviour{
      public Addition(Agent a, int durée){
         super(a, durée);
      }// Constructeur Addition
```