Université Constantine 2

Faculté des NTIC

Département d'Informatique Fondamentale et ses Applications

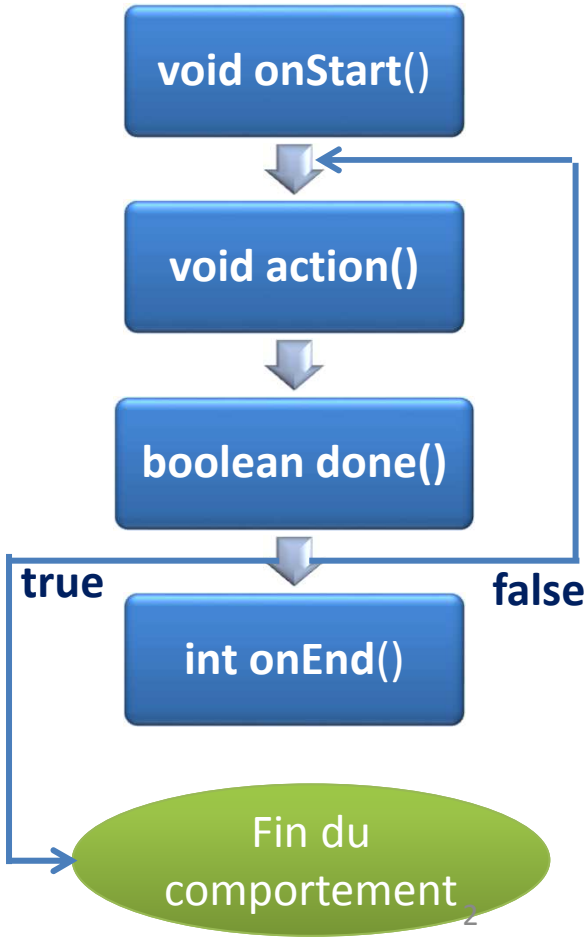
1ère Année Master Réseaux et Systèmes Distribués
TP Algorithmes Distribués (ALDI)

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

Les comportements

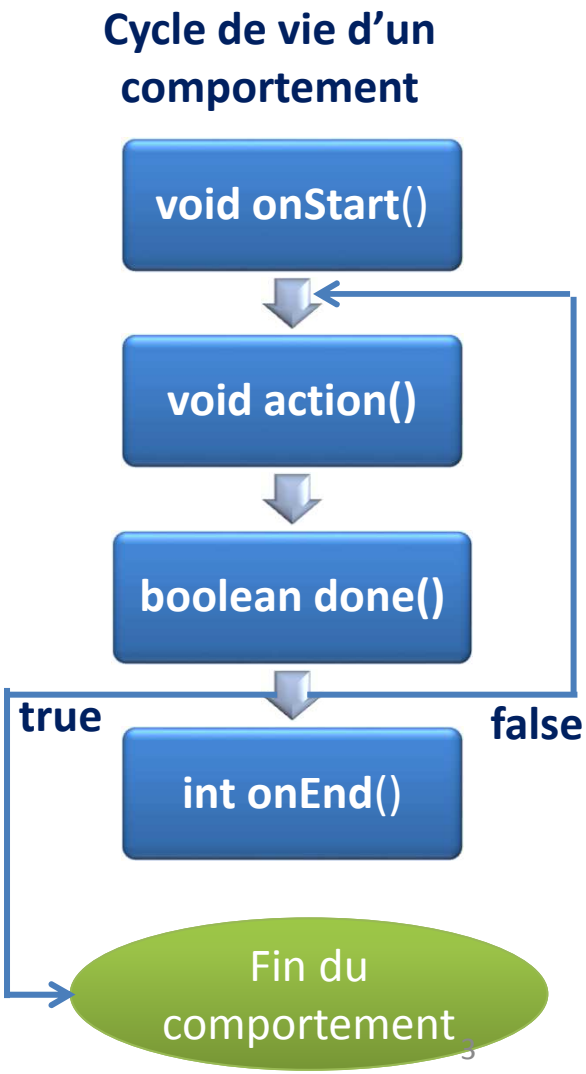
- Dans la plateforme **JADE**, un agent possède un ou plusieurs comportements (Behaviours) qui définissent ses actions
- Un comportement hérite de la classe **jade.core.behaviours**
- Chaque comportement doit implémenter au moins les deux méthodes :
  - ✓ void **action**() : désigne les opérations à 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**() .

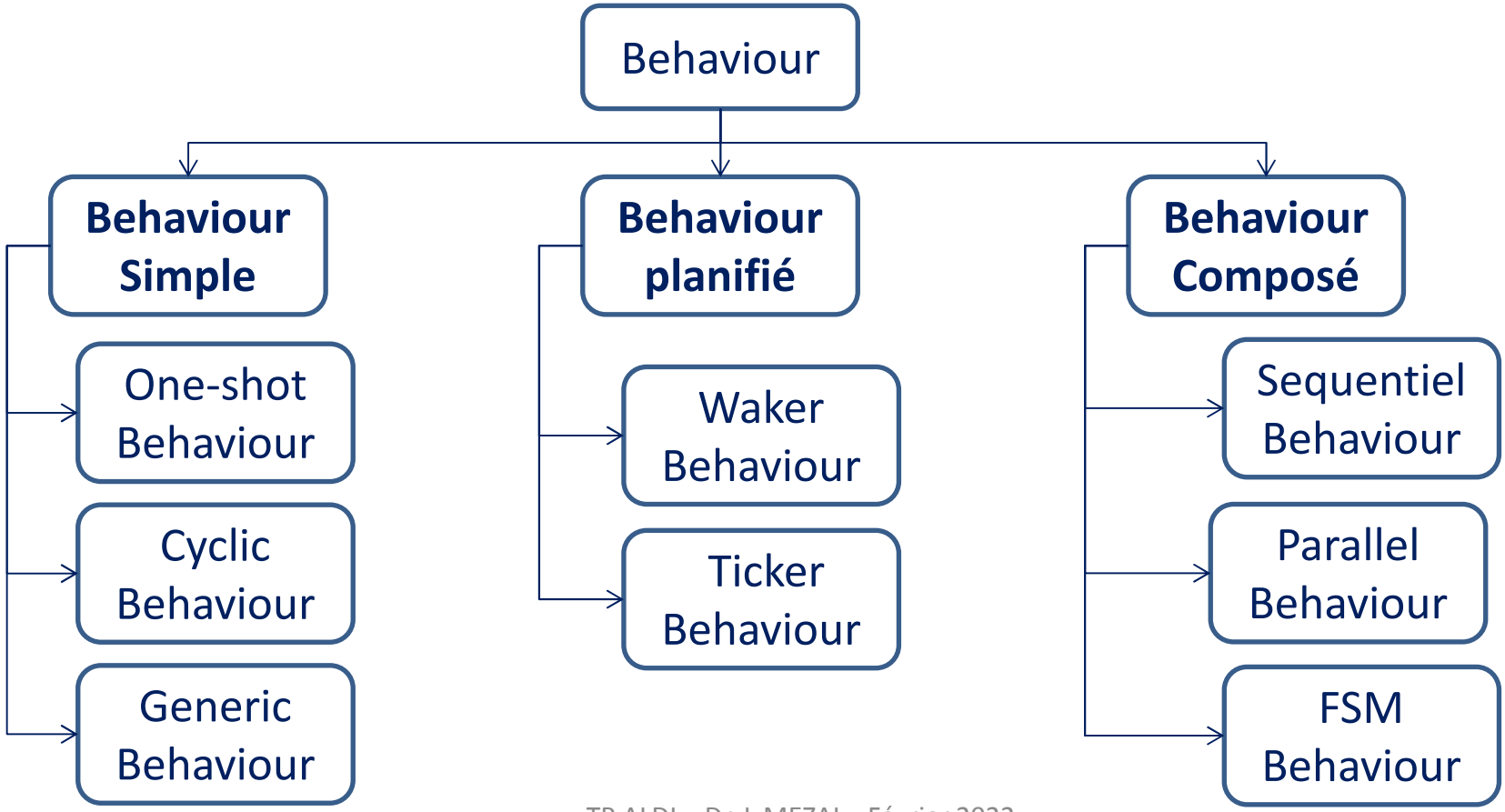


Les comportements

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 {
      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
```

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: (suite)

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

Exécuter le programme en changeant la valeur de la période

Les comportements

Les comportements Composés

- **SequentielBehaviour**
  - ✓C'est un comportement composé de plusieurs sous comportements
  - ✓Il commence par exécuter le premier sous comportement
  - ✓Lorsque celui-là termine son exécution, il passe au prochain sous comportement, et ainsi de suite.
  - ✓Les sous comportements sont ajoutés au sequentielBehaviour par la méthode addSubBehaviour().
  - ✓L'ordre de l'ajout détermine l'ordre d'exécution.

Exemple:

```
public class Calculateur extends Agent {
   protected void setup(){
      System.out.println("Je suis l'agent : "+getLocalName());
      SequentialBehaviour ComportSeq = new SequentialBehaviour();
      ComportSeq.addSubBehaviour(new Addition());
      ComportSeq.addSubBehaviour(new Soustraction());
      ComportSeq.addSubBehaviour(new Produit());
      addBehaviour(ComportSeq);
   }//setup
```

Les comportements

Les comportements Composés

- **SequentielBehaviour**

Exemple: (suite)

```
public class Addition extends Behaviour{
   public void action() {
      int c;
      System.out.println("le 1er sous-comportement");
      int a = (int)(Math.random() * 100);
      int b = (int)(Math.random() * 100);
      c = a + b;
      System.out.println("Je suis l'agent "+getLocalName()+": j'ai calculé :"+a+"+"+b+"=" +c);
   } //action
   public boolean done(){
        return c == 100;
   } //done
}// class Addition
```

Les comportements

Les comportements Composés

- **SequentielBehaviour**

Exemple: (suite)

```
public class Soustraction extends Behaviour{
   int c;
   public void action() {
      System.out.println("le 2ième sous-comportement");
      int a = (int)(Math.random() * 100);
      int b = (int)(Math.random() * 100);
      c = a - b;
      System.out.println("Je suis l'agent "+getLocalName()+": j'ai calculé :"+a+"-"+b+"=" +c);
   } //action
   public boolean done(){
         return c < 0;
   }//done
}// class Soustraction
```

Les comportements

Les comportements Composés

- **SequentielBehaviour**

Exemple: (suite)

```
   public class Produit  extends OneShotBehaviour{
      public void action() {
         System.out.println("le 3ième sous-comportement");
         int a = (int)(Math.random() * 100);
         int b = (int)(Math.random() * 100);
         int c = a  * b;
         System.out.println("Je suis l'agent "+getLocalName()+": j'ai calculé :"+a+"*"+b+"=" +c);
      } // action
   }// class Produit
}//class Calculateur
```

<u>Question :</u>

Le 1er sous-comportement peut-il être de type CyclicBehavoir?

Les comportements

Les comportements Composés

- **ParallelBehaviour**
  - ✓C'est une instance de la classe : jade.core.behaviours.ParallelBehaviour.
  - ✓Il permet d'exécuter plusieurs sous comportements en parallèle.
  - ✓L'ajout d'un sous comportement fait par la méthode addSubBehaviour() .
  - ✓Si on veut que le parallelBehaviour se termine dès qu'un de ses sous comportements termine son exécution alors on doit passer à son constructeur l'argument **WHEN_ANY**.
  - ✓Pour attendre la fin de tous les sous comportements on doit lui passer l'argument **WHEN_ALL**.

Exemple:

```
public class Calculateur extends Agent{
  protected void setup(){
    System.out.println("Je suis l'agent : "+getLocalName());
    ParallelBehaviour ComportParallel = new
                    ParallelBehaviour(ParallelBehaviour.WHEN_ANY);
    ComportParallel.addSubBehaviour(new Addition());
    ComportParallel.addSubBehaviour(new Soustraction());
    ComportParallel.addSubBehaviour(new Produit());
    addBehaviour(ComportParallel);
  }//setup
```

Remarque :
L'implémentation de ces comportements est identique à l'exemple du SequentielBehaviour

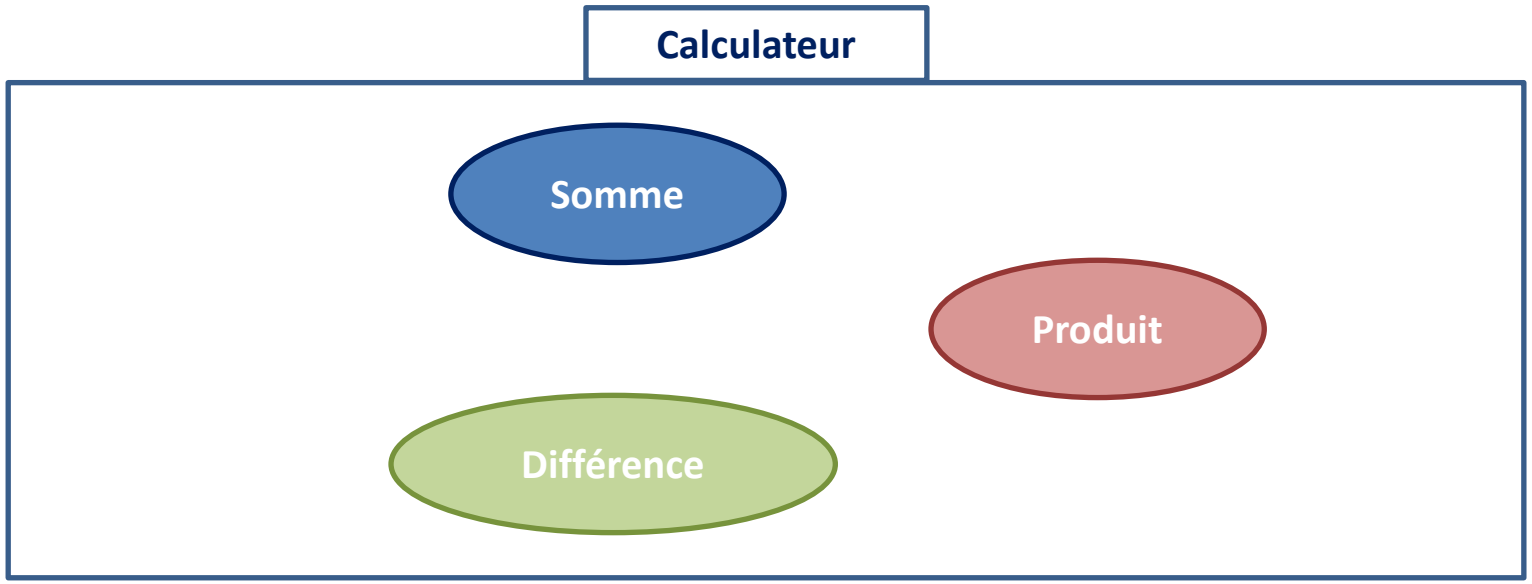
Les comportements

Les comportements Composés

- **FSMBehaviour**
  - ✓Le FSMBehaviour (Finite State Machine Behaviour) est une instance de la classe : **jade.core.behaviours.FSMBehaviour.**
  - ✓Il permet d'implémenter un automate à états finis dont chaque état correspond à l'exécution d'un sous comportement.

Exemple :

Nous considérons un agent Calculateur qui utilise un FSMBehaviour. Il est composé de trois états : somme, produit et différence. Le fonctionnement de ce comportement est décrit dans ce qui suit :



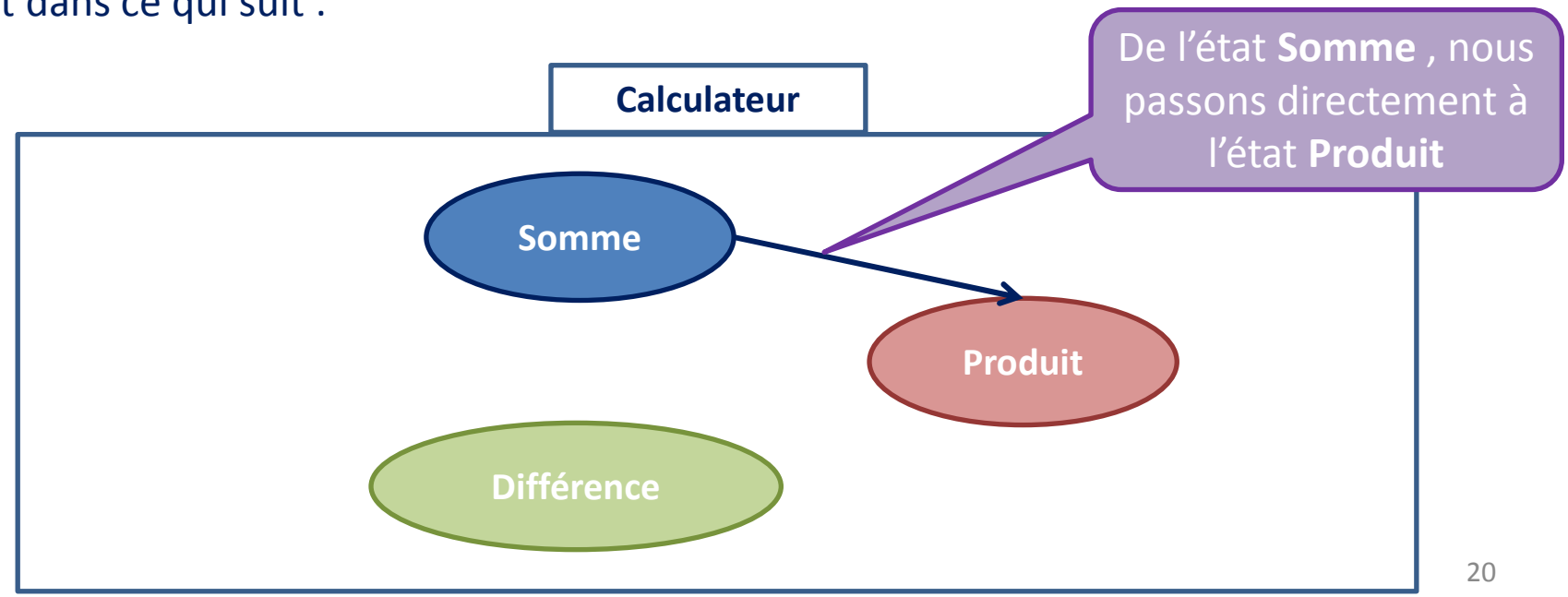
Les comportements

Les comportements Composés

- **FSMBehaviour**
  - ✓Le FSMBehaviour (Finite State Machine Behaviour) est une instance de la classe : **jade.core.behaviours.FSMBehaviour.**
  - ✓Il permet d'implémenter un automate à états finis dont chaque état correspond à l'exécution d'un sous comportement.

Exemple :

Nous considérons un agent Calculateur qui utilise un FSMBehaviour. Il est composé de trois états : somme, produit et différence. Le fonctionnement de ce comportement est décrit dans ce qui suit :



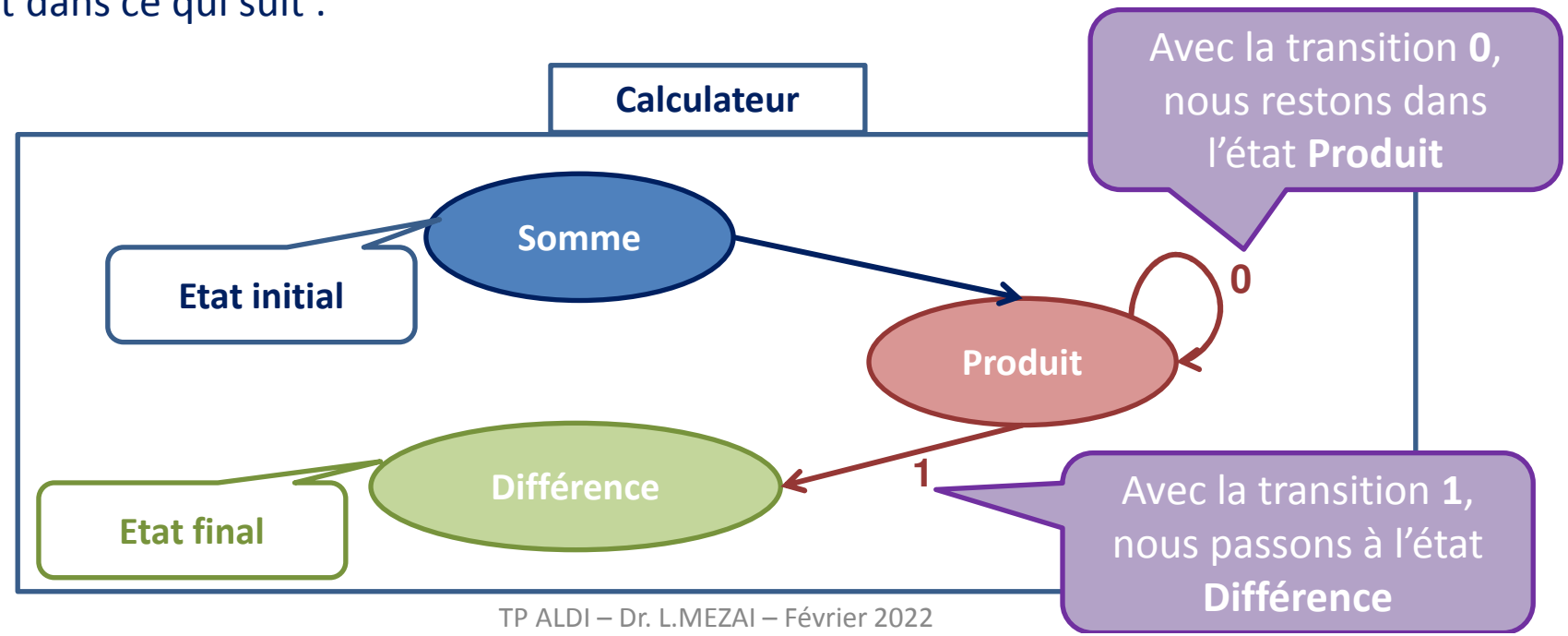
Les comportements

Les comportements Composés

- **FSMBehaviour**
  - ✓Le FSMBehaviour (Finite State Machine Behaviour) est une instance de la classe : **jade.core.behaviours.FSMBehaviour.**
  - ✓Il permet d'implémenter un automate à états finis dont chaque état correspond à l'exécution d'un sous comportement.

Exemple :

Nous considérons un agent Calculateur qui utilise un FSMBehaviour. Il est composé de trois états : somme, produit et différence. Le fonctionnement de ce comportement est décrit dans ce qui suit :



Les comportements

Les comportements Composés

- **FSMBehaviour**

La classe **FSMBehaviour** offre un ensemble de méthodes que nous devons utiliser.

✓L'ajout de l'état initial (il n'existe qu'un seul état initial) se fait par la méthode :

registerFirstState (Behaviour NomBehaviour, String NomEtat);

✓L'ajout d'un état se fait par la méthode :

registerState (Behaviour NomBehaviour, String NomEtat) ;

✓L'ajout d'un état final (il est possible d'en avoir plusieurs) se fait par la méthode :

registerLastState (Behaviour NomBehaviour, String NomEtat);

✓L'ajout d'une transition se fait par la méthode :

registerTransition(String NomEtat1, String NomEtat2, int ValeurTransition);

✓L'ajout d'une transition par défaut se fait par la méthode :

registerDefaultTransition(String NomEtat1, String NomEtat2)

Les comportements

Les comportements Composés

- **FSMBehaviour**

  Exemple détaillé:

```
public class Calculateur extends Agent{
   protected void setup(){
      System.out.println("Je suis l'agent : "+getLocalName());
      FSMBehaviour ComportCalcul = new FSMBehaviour(this) ;
      //définition des états
      ComportCalcul.registerFirstState (new Addition(),"Somme");
      ComportCalcul.registerState(new Multiplication(),"Produit");
      ComportCalcul.registerLastState(new Soustraction(),"Difference");
      //définition des transitions
      ComportCalcul.registerDefaultTransition("Somme","Produit");
      ComportCalcul.registerTransition("Produit","Produit",0);
      ComportCalcul.registerTransition("Produit","Difference",1);
      addBehaviour(ComportCalcul);
   }
```

Les comportements

Les comportements Composés

- **FSMBehaviour**

  Exemple détaillé:

```
private class Addition extends OneShotBehaviour{
   public void action() {
      System.out.println("Agent"+getLocalName()+": Etat Initial”);
      System.out.println("Agent"+getLocalName()+":execution de l'etat:"+
      getBehaviourName());
      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);
   }
}
```

Les comportements

Les comportements Composés

- **FSMBehaviour**

  Exemple détaillé:

```
private class Multiplication extends OneShotBehaviour{
   int  c;
   public void action() {
      System.out.println("Agent"+getLocalName()+":execution de l'etat:"+
      getBehaviourName());
      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);
   }
   public int onEnd(){
      int valTransition = (int) (Math.random() * 2); //générer une valeur qui sera égale à 0
      return valTransition;                          // ou bien elle sera égale à 1
   }
}
```

Les comportements

Les comportements Composés

- **FSMBehaviour**

  Exemple détaillé:

```
  private class Soustraction extends OneShotBehaviour{
    public void action() {
      System.out.println("Agent"+getLocalName()+":Etat Finale”);
      System.out.println("Agent"+getLocalName()+":execution de l'etat:"+getBehaviourName());
      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);
      //arrêt de l’agent
      myAgent.doDelete();
    }// action
  }// Soustraction
} //Calculateur
```