Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Science and Technology Houari Boumediene

Department of AI & DS

Field: Computer Science

Specialisation:

Intelligent Computer Systems

REPORT TPs: Agent Technology

Produced by:

Ferrah Anissa 202033018072

Table des matières

1. Introduction :

The report provides a comprehensive exploration of two fundamental subjects: Expert Systems (ES) and the JADE (Java Agent Development Framework) platform. It begins with a detailed examination of expert systems, which are hailed as the pinnacle of AI for encapsulating human expertise in specific domains. Delving into their architecture, methodologies, and algorithms, the report sheds light on forward chaining and backward chaining approaches through practical examples and Java implementations.

Moving on to Chapter 2, the report introduces the JADE platform, an open-source framework facilitating the development of autonomous software agents in Java. Core concepts such as agents, platforms, agent containers, behaviors, messages, and multi-agent platforms are elucidated, accompanied by hands-on exercises that guide readers through agent creation, argument passing mechanisms, communication, and behaviors within the JADE framework.

In the subsequent chapter, the focus shifts to communication within JADE, encompassing various communication mechanisms from point-to-point messaging to broadcast communication. Through practical examples, readers gain insights and practical skills to develop efficient multi-agent applications.

Concluding reflections highlight the significance of backward chaining and forward chaining in AI problem-solving, emphasizing their role in developing intelligent systems and their impact on problem diagnosis and resolution. Additionally, the report underscores the importance of behaviors in Chapter 4, emphasizing their role in shaping agent actions and interactions.

Overall, the report serves as a comprehensive resource, providing readers with a deep understanding of expert systems, the JADE platform, behaviors, and their applications in artificial intelligence and multi-agent systems development.

2. Chapter 1: Expert systems (ES)

2.1. Introduction :

Expert systems (ES) are described as the pinnacle of AI, embodying human expertise within specific domains. They comprise computer programs designed to emulate human problem-solving capabilities and are used across various fields. The architecture of an ES typically includes an inference engine, knowledge base, and user interface. Forward chaining and backward chaining are discussed as two major approaches in AI, each offering distinct methods for problem-solving. Forward chaining progresses from initial facts to conclusions, while backward chaining starts with a goal and works backward to find necessary facts. These techniques are essential for logic and expert systems, contributing to automated reasoning and inference. The choice between them depends on the problem's nature and specific goals of the intelligent system.

2.2. Forward Chaining :

Forward chaining is a method used in artificial intelligence and expert systems to make decisions or draw conclusions based on an initial set of facts or data. It starts with the available information and progresses through a series of logical steps to arrive at a conclusion or solution. This approach is particularly useful when the system has access to a large amount of data and needs to process it to reach a specific goal. In forward chaining, the system begins with known facts and uses rules or algorithms to infer additional information until it reaches a desired outcome. This method is commonly used in systems where the goal is to analyze data or detect patterns to make predictions or recommendations.

2.2.1. Forward Chaining Algorithm:

```
Parameters: the fact (to demonstrate)
if fact in BF then
   res = "SUCCESS"
else
   nonTriggeredRules = BR ;rulesToConsider = BR ; res = "FAILURE"
 while rulesToConsider !={} and res != "SUCCESS" do
# Choose a rule to consider
   r = choose(rulesToConsider)
  # Remove the chosen rule from rulesToConsider
rulesToConsider = rulesToConsider - {r}
# Check if all premises are in the base of facts
    if all(p in BF , p in premise(r)) then
# Update the base of facts with the conclusion of the rule
 BF = BF+{conclusion(r)}
 # Remove the triggered rule from non-triggered rules
nonTriggeredRules = nonTriggeredRules - {r}
rulesToConsider = nonTriggeredRules

  if conclusion(r) == fact then
res = "SUCCESS End if End if End while End if
Return(res)
```

2.2.2. Implementation and Application of Forward Chaining Algorithm in Java for Expert Systems with Example:

2.2.2.1. Example of using the Forward Chaining algorithm to solve a problem in an expert system:

Rules:

Rule 1: If A and B are true, then F is true.
Rule 2: If F and H are true, then I is true.
Rule 3: If D, H, and G are true, then A is true.
Rule 4: If O and G are true, then H is true.
Rule 5: If E and H are true, then B is true.
Rule 6: If G and A are true, then B is true.
Rule 7: If G and H are true, then P is true.
Rule 8: If G and H are true, then O is true.
Rule 9: If D, O, and G are true, then J is true.

Facts to prove: **I**
The initial facts: **D, O, G**

2.2.2.2. Application of Forward Chaining Algorithm:

```java
package Tp1;
import java.util.ArrayList;
import java.util.Arrays;

class forward{
    public static boolean ForwardChaining(Rule[] bdr, ArrayList<String> bdf, ArrayList<String> FP) {

        boolean res = false;

        // Tant qu'il reste des faits à prouver et que le résultat n'est pas encore trouvé
        while (!FP.isEmpty() && !res) {
            String fact = FP.get(0);
            FP.remove(0);

            // Si le fait à prouver est déjà dans la base de faits, on passe au suivant
            if (isFactInBDF(fact, bdf)) {
                continue;
            }

            // Initialisation des règles non déclenchées et des règles à considérer
            ArrayList<Rule> untriggeredRules = new ArrayList<>(Arrays.asList(bdr));
            ArrayList<Rule> rulesToConsider = new ArrayList<>(Arrays.asList(bdr));

            // Tant qu'il reste des règles à considérer et que le résultat n'est pas encore trouvé
            while (!rulesToConsider.isEmpty() && !res) {
                Rule rule = chooseRule(rulesToConsider);
                rulesToConsider.remove(rule);

                // Si toutes les prémisses de la règle sont dans la base de faits
                if (containsAll(rule.P, bdf)) {
                    // Ajouter la conclusion de la règle à la base de faits
                    bdf.addAll(rule.C);

                    // Supprimer la règle des règles non déclenchées
                    untriggeredRules.remove(rule);

                    // Mettre à jour les règles à considérer
                    rulesToConsider = new ArrayList<>(untriggeredRules);

                    // Si la conclusion de la règle est le fait à prouver, alors succès
                    if (rule.C.contains(fact)) {
                        res = true;
                    }
                }
            }
        }
        return res;
    }

    public static Rule chooseRule(ArrayList<Rule> rules) {
        // Pour simplifier, on choisit simplement la première règle dans la liste
        return rules.get(0);
    }

    public static boolean containsAll(ArrayList<String> subset, ArrayList<String> superset) {
        return superset.containsAll(subset);
    }

    public static boolean isFactInBDF(String fact, ArrayList<String> bdf) {
        return bdf.contains(fact);
    }
```

```java
    public static void main(String[] args) {
        Rule[] bdr = new Rule[9];

        // Base de règles
        ArrayList<String> P = new ArrayList<>(Arrays.asList("A", "B"));
        ArrayList<String> C = new ArrayList<>(Arrays.asList("F"));
        bdr[0] = new Rule(0, P, C);

        P = new ArrayList<>(Arrays.asList("F", "H"));
        C = new ArrayList<>(Arrays.asList("I"));
        bdr[1] = new Rule(1, P, C);

        P = new ArrayList<>(Arrays.asList("D", "H", "G"));
        C = new ArrayList<>(Arrays.asList("A"));
        bdr[2] = new Rule(2, P, C);

        P = new ArrayList<>(Arrays.asList("O", "G"));
        C = new ArrayList<>(Arrays.asList("H"));
        bdr[3] = new Rule(3, P, C);

        P = new ArrayList<>(Arrays.asList("E", "H"));
        C = new ArrayList<>(Arrays.asList("B"));
        bdr[4] = new Rule(4, P, C);

        P = new ArrayList<>(Arrays.asList("G", "A"));
        C = new ArrayList<>(Arrays.asList("B"));
        bdr[5] = new Rule(5, P, C);

        P = new ArrayList<>(Arrays.asList("G", "H"));
        C = new ArrayList<>(Arrays.asList("P"));
        bdr[6] = new Rule(6, P, C);

        P = new ArrayList<>(Arrays.asList("G", "H"));
        C = new ArrayList<>(Arrays.asList("O"));
        bdr[7] = new Rule(7, P, C);

        P = new ArrayList<>(Arrays.asList("D", "O", "G"));
        C = new ArrayList<>(Arrays.asList("J"));
        bdr[8] = new Rule(8, P, C);

        // Faits à prouver
        ArrayList<String> FP = new ArrayList<>(Arrays.asList("I"));

        // Base de faits
        ArrayList<String> bdf = new ArrayList<>(Arrays.asList("D", "O", "G"));



        // Premier appel
        boolean test = ForwardChaining(bdr, bdf, FP);

        // Affichage du résultat
        System.out.println(test);

    }


}

```

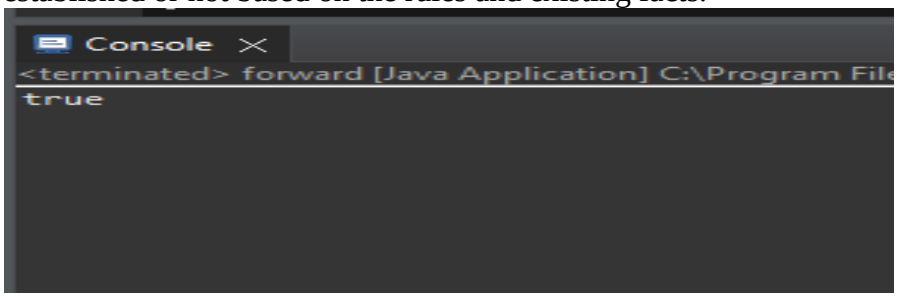
2.2.2.3. Explanation of Forward Chaining Algorithm for Expert Systems:

- **Initialization**: The code starts by defining a set of logical rules, each rule having premises and conclusions. It also specifies facts to be proven and a base of existing facts.

```java
package Tp1;

import java.util.ArrayList;

public class Rule {
    boolean State;
    ArrayList<String> P;
    ArrayList<String> C;
    int name;

    public Rule(int i, ArrayList<String> a1, ArrayList<String> a2) {
        name = i;
        State = true;
        P = a1;
        C = a2;

    }
}

```

- **Algorithm**: The forward chaining algorithm begins with a list of facts to be proven. At each iteration, it selects a fact to be proven and checks whether the premises of certain rules are satisfied by the already known facts. If so, the conclusion of the rule is added to the base of facts, and the process continues until the fact to be proven is established or all possibilities are exhausted.

- **Result Display**: Finally, the program displays whether the facts to be proven have been established or not based on the rules and existing facts.



2.3.Back Ward Chaining :

2.3.1. Backward Chaining Algorithm:

```
Function BackwardChaining
Parameters: in BR, in BF, in goalList.
if isEmpty(goalList) then
   result = SUCCESS
else
   if takegoal (first(goalList)) then
      result =BackwardChaining(rest(goalList))
   else
      result =FAILURE
   end if
end if
return result
```

Function takegoal:

```
Function takegoal
Parameters: in BR, in BF, in goal.
if goal in BF then
   result =SUCCESS
else
   rules = BR
   result = FAILURE

   while !isEmpty(rules) and result != SUCCESS:
      r = ruleChoice(rules)
      rules = rules - {r}
      if goal in conclusion(r) then
         result = BackwardChaining(BR, BF, premise(r))
end if
end while
return result
end if
```

2.3.2. Implementation and Application of Backward Chaining Algorithm in Java for Expert Systems with Example:

2.3.2.1. Application of Backward Chaining Algorithm with Example:

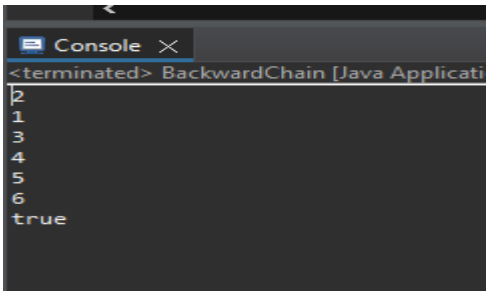
```java
package Tp1;

import java.util.ArrayList;
import java.util.Arrays;

public class backward {
    public static boolean BackwardChaining(Rule[] bdr, ArrayList<String> bdf, ArrayList<String> FP) {
        boolean res;
        if (FP.isEmpty()) { res=true;}
        else {
            if (takeGoal(bdr,bdf,FP.get(0))) {
                bdf.add(FP.get(0));
                res = BackwardChaining(bdr,bdf,rest(FP));
            }
            else {
                res= false;
            }
        }
        return res;
    }

    public static boolean takeGoal(Rule[] bdr, ArrayList<String> bdf, String goal) {
        boolean res;
        if (bdf.contains(goal)) {
            res= true;
        }
        else {
            res= false;
            //liste des conflits
            ArrayList<Rule> RA = new ArrayList<Rule>();

            for (int i=0; i<bdr.length; i++) {
                if (bdr[i].C.contains(goal)) {
                    RA.add(bdr[i]);
                }
            }
            while(!RA.isEmpty() && res!=true) {
                Rule r = RA.get(0);
                System.out.println(r.name+1);
                RA.remove(0);
                res= BackwardChaining(bdr,bdf,r.P);
            }
        }


        return res;
    }

    //fonction rest
    public static ArrayList<String> rest(ArrayList<String> a) {
        ArrayList<String> r = new ArrayList<String>();
        for (int i=1; i<a.size();i++) {
            r.add(a.get(i));
        }
        return r;
    }
```

```java
    public static void main(String[] args) {
        Rule[] bdr = new Rule[9];

        //base de regles et un tableau
        ArrayList<String> P = new ArrayList<> (Arrays.asList("A","B"));
        ArrayList<String> C = new ArrayList<> (Arrays.asList("F"));
        bdr[0]= new Rule(0,P,C);

        P = new ArrayList<> (Arrays.asList("F","H"));
        C = new ArrayList<> (Arrays.asList("I"));
        bdr[1]= new Rule(1,P,C);

        P = new ArrayList<> (Arrays.asList("D","H","G"));
        C = new ArrayList<> (Arrays.asList("A"));
        bdr[2]= new Rule(2,P,C);

        P = new ArrayList<> (Arrays.asList("O","G"));
        C = new ArrayList<> (Arrays.asList("H"));
        bdr[3]= new Rule(3,P,C);

        P = new ArrayList<> (Arrays.asList("E","H"));
        C = new ArrayList<> (Arrays.asList("B"));
        bdr[4]= new Rule(4,P,C);

        P = new ArrayList<> (Arrays.asList("G","A"));
        C = new ArrayList<> (Arrays.asList("B"));
        bdr[5]= new Rule(5,P,C);

        P = new ArrayList<> (Arrays.asList("G","H"));
        C = new ArrayList<> (Arrays.asList("P"));
        bdr[6]= new Rule(6,P,C);

        P = new ArrayList<> (Arrays.asList("G","H"));
        C = new ArrayList<> (Arrays.asList("O"));
        bdr[7]= new Rule(7,P,C);

        P = new ArrayList<> (Arrays.asList("D","O","G"));
        C = new ArrayList<> (Arrays.asList("J"));
        bdr[8]= new Rule(8,P,C);

        //fact to prove ( GOAL)
        ArrayList<String> FP = new ArrayList<> (Arrays.asList("I"));

        //facts Base
        ArrayList<String> bdf = new ArrayList<> (Arrays.asList("D","O","G"));

        //first call
        boolean test = BackwardChaining(bdr,bdf,FP);

        //print result
        System.out.println(test);
    }

}

```

2.3.2.2. Explanation of back ward Chaining Algorithm for Expert Systems:

Here is a general explanation of the code:

- **Rule Class**: Represents a rule with a list of premises (P) and a list of conclusions (C). Each rule has an associated name (name).
- **BackwardChaining Function**: This function is the entry point for backward chaining. It takes an array of rules (bdr), a base of facts (bdf), and a set of facts to prove (FP) as parameters. It returns a boolean indicating whether the goal can be proven or not.
- **takeGoal Function**: This function is used to check if a specific goal can be deduced from the current base of facts (bdf). If the goal is already in the base of facts, the function returns true. Otherwise, it looks for rules whose conclusion matches the goal and attempts to prove the premises of those rules recursively.
- **rest Function**: A utility function that returns a sublist from the element at index 1 to the end of the list.
- **Result Display**:



The program creates a set of rules (bdr), a base of facts (bdf), and a set of facts to prove (FP). It then calls the BackwardChaining function with these parameters and displays the result.

2.4. Conclusion:

Backward chaining and forward chaining are two distinct approaches in artificial intelligence for solving logical deduction problems. Backward chaining is particularly effective when the goal is clearly defined, as it starts by identifying the goal to be achieved and then recursively works backward through logical rules to prove the necessary premises. This approach is often favored in expert systems where problem diagnosis is crucial. On the other hand, forward chaining is employed when the system has a large number of rules and initial data, and it needs to progressively deduce conclusions. This method is suitable for control systems where actions are based on available data. In summary, the choice between backward chaining and forward chaining depends on the characteristics of the problem to be solved, with each method offering its own advantages in specific contexts.

3. Chapter 2 : JADE introduction

3.1. Introduction:

JADE (Java Agent développement Framework) est une plate-forme open source qui facilite le développement d'agents logiciels en Java. Les agents sont des entités autonomes capables de percevoir leur environnement, de prendre des décisions et d'agir en conséquence pour atteindre leurs objectifs. JADE offre une infrastructure robuste et flexible pour créer, déployer et gérer des agents dans divers contextes d'application. Les principaux concepts de JADE comprennent les agents, les plateformes, les conteneurs d'agents, les comportements, les messages et les plates-formes multi-agents. Cette plateforme permet le développement de systèmes autonomes et intelligents dans des domaines tels que l'automatisation industrielle, la gestion des systèmes d'information, et bien d'autres encore.

3.2. JADE Agent Creation and Launching: Passing argument:

1/Create an agent in JADE platform that displays the message « Hello world » followed by its name.

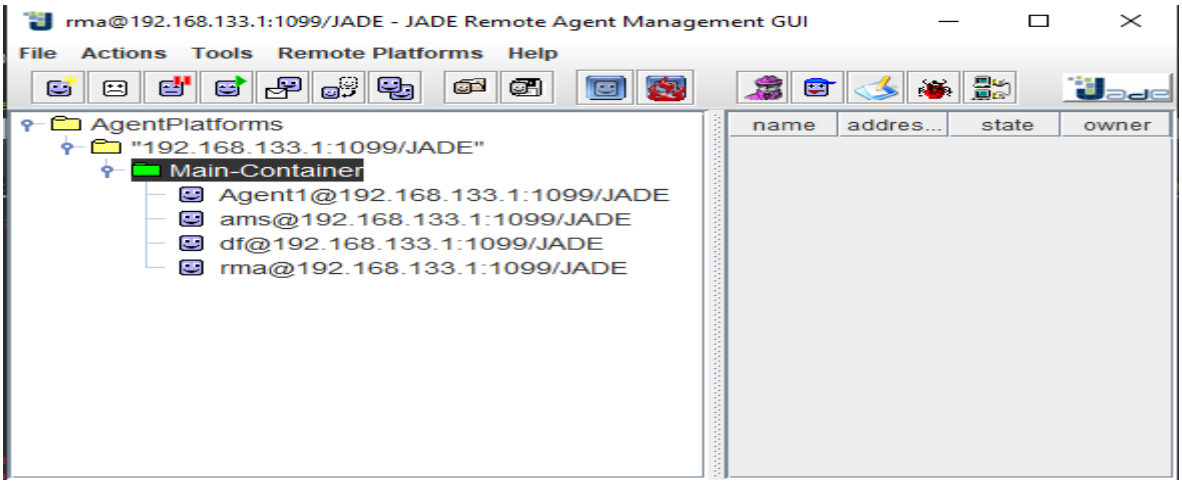
```java
package Tp2;

import jade.core.Agent;

public class Agent1 extends Agent {

    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        super.setup();
        System.out.println("Hello world, My name is "+getLocalName());
    }
}
```

2/ the steps for launching one agent.

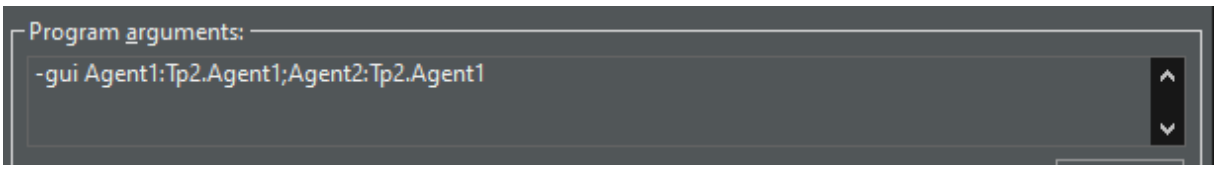
after execution :

```
Hello world, My name is Agent1
```

A new agent has been inserted into the JADE platform.



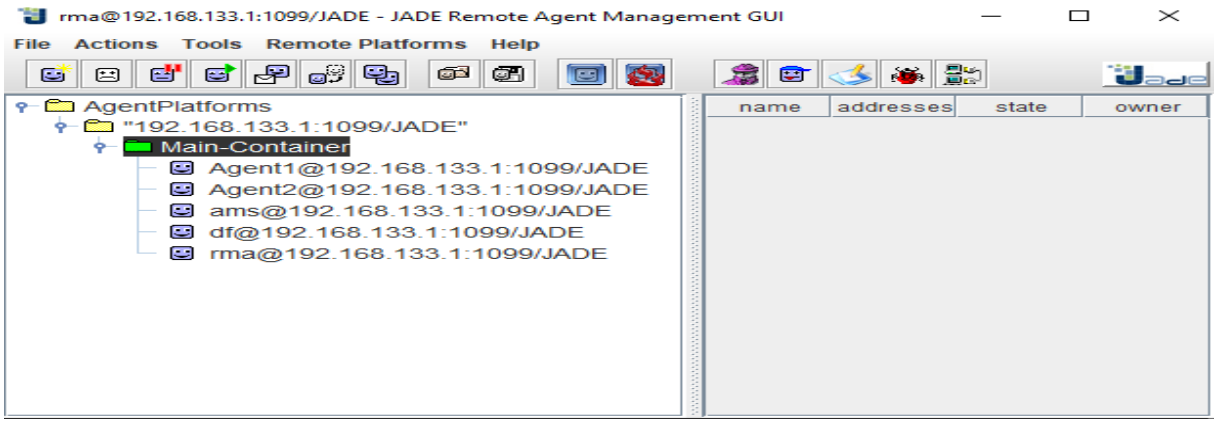
3/ Give the steps for launching two agents.



after execution :

```
Hello world, My name is Agent2
Hello world, My name is Agent1
```

Two new agents have been inserted into the JADE platform.

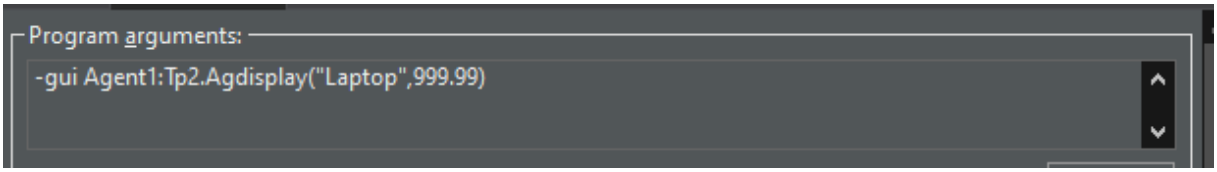


3.3.Developing JADE Agents with Argument Passing (Product Selling):

1/ Develop a Jade agent class, which receives data as arguments and displays it.

```java
package Tp2;
import jade.core.Agent;

public class Agdisplay extends Agent {

    float price;
    String name;
    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        super.setup();//pour lancer les traitements de l'agent
        Object[] args = getArguments();
        if(args!=null) {
            name = (String) args[0];
            price = Float.valueOf((String) args[1]).floatValue();
        }
        System.out.println("In Agent : "+getLocalName()+" Sell product "+name+" at price "+price);

    }

}
```

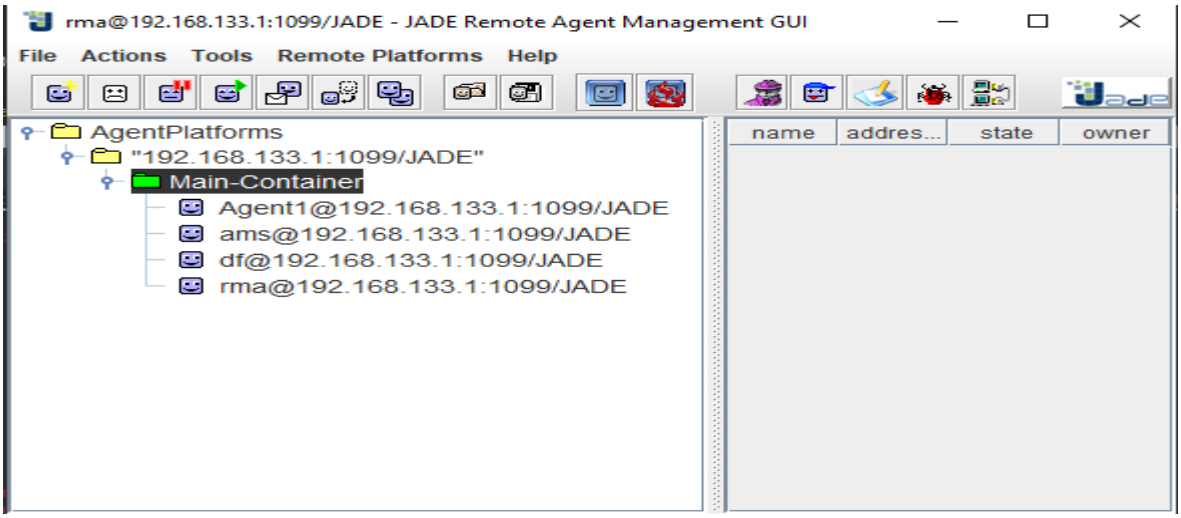
2/ Give the steps for launching an agent with passing arguments.



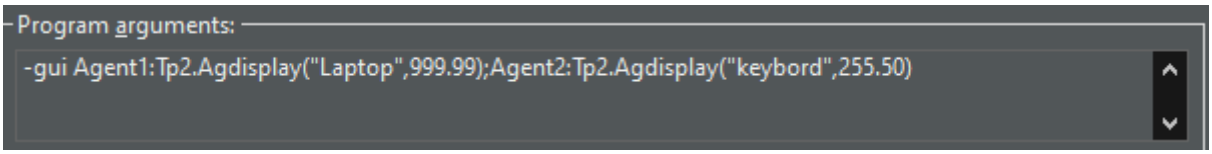
after execution :

```
In Agent : Agent1 Sell product Laptop at price 999.99
```

A new agent has been inserted into the JADE platform.



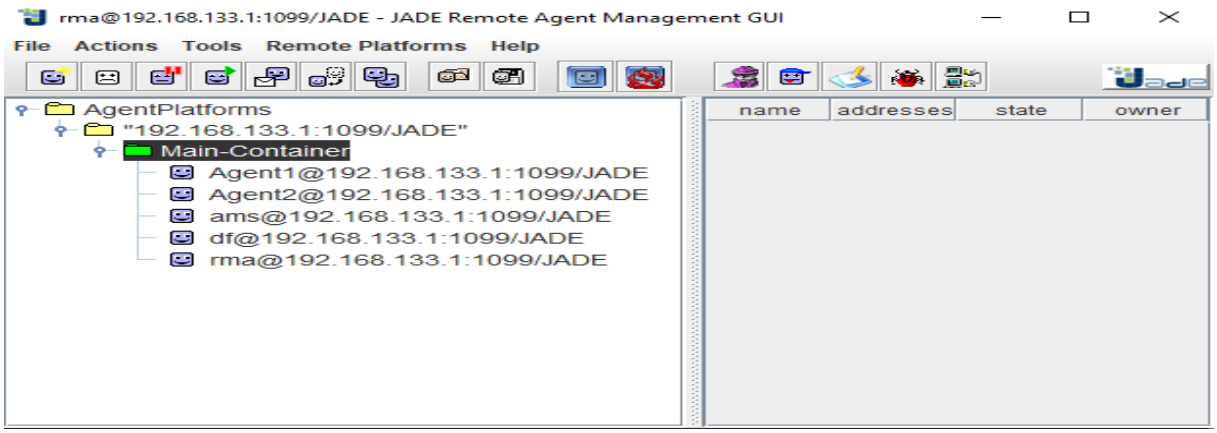
3/ Give the steps for launching two agents with passing arguments.



after execution :

```
In Agent : Agent2 Sell product keybord at price 255.5
In Agent : Agent1 Sell product Laptop at price 999.99
```

Two new agents have been inserted into the JADE platform.



3.4.JADE Agent Creation and Launching: Program java:

1/Create an agent in JADE platform that displays the message « Hello world » followed by its name.

```java
package Tp2;

import jade.core.Agent;

public class Agent1 extends Agent {

    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        super.setup();
        System.out.println("Hello world, My name is "+getLocalName());
    }
}
```

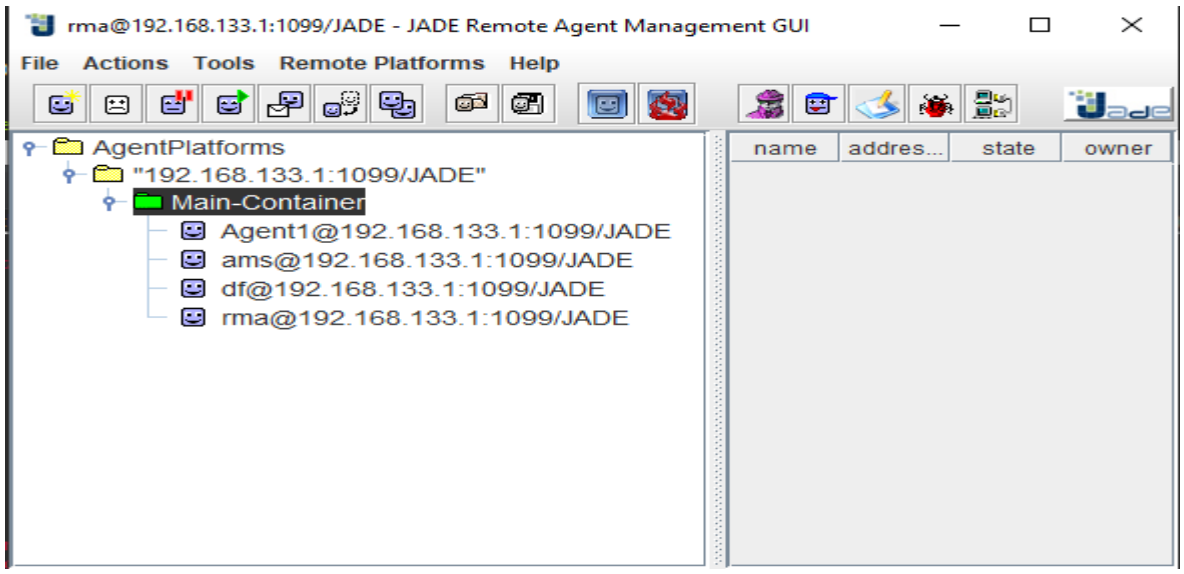
2/ the steps for launching one agent by program java

```java
package Tp2;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;

public class Agent_launcher {

    public static void main(String[] args) {

        try {
            Runtime rt = Runtime.instance();
            ProfileImpl p = new ProfileImpl();
            p.setParameter(Profile.LOCAL_HOST,"localhost");
            p.setParameter(Profile.LOCAL_PORT,"1099");
            p.setParameter(Profile.GUI,"true");
            ContainerController mc = rt.createMainContainer(p);
            AgentController ag1 = mc.createNewAgent("Agent1", "Tp2.Agent1", null);

            ag1.start();

        } catch (StaleProxyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
```

after execution :

```
-----------------------------------------
Hello world, My name is Agent1
```

A new agent has been inserted into the JADE platform.



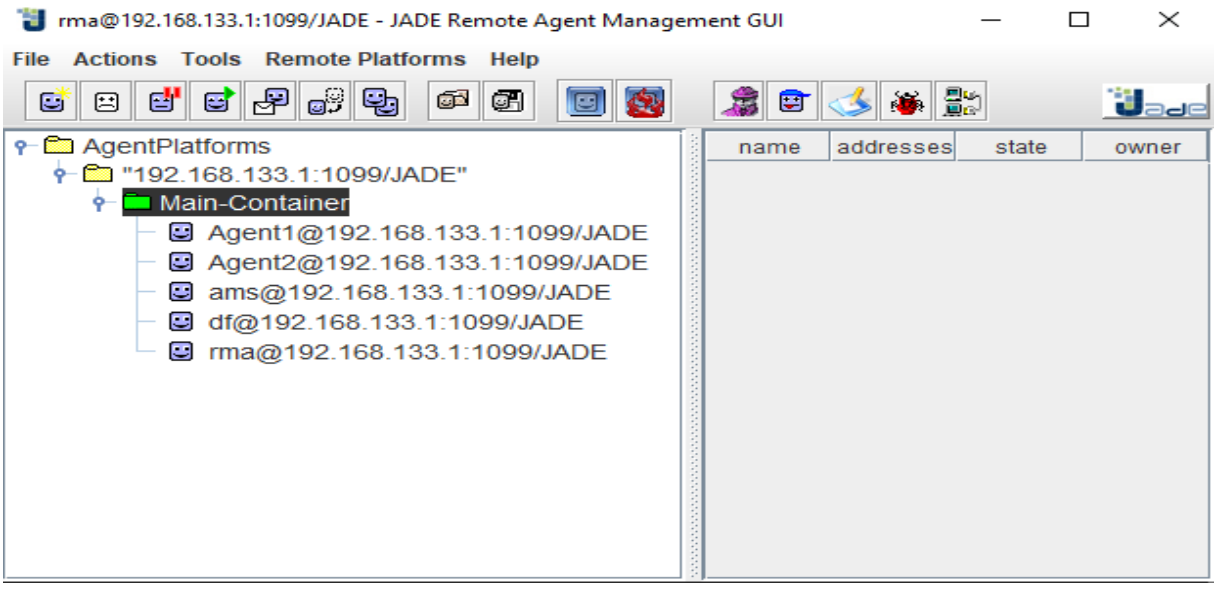
3/ Give the steps for launching two agents by program java

```java
package Tp2;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;

public class Agent_launcher {

    public static void main(String[] args) {

        try {
            Runtime rt = Runtime.instance();
            ProfileImpl p = new ProfileImpl();
            p.setParameter(Profile.LOCAL_HOST,"localhost");
            p.setParameter(Profile.LOCAL_PORT,"1099");
            p.setParameter(Profile.GUI,"true");
            ContainerController mc = rt.createMainContainer(p);
            AgentController ag1 = mc.createNewAgent("Agent1", "Tp2.Agent1", null);

            ag1.start();
            AgentController ag2 = mc.createNewAgent("Agent2", "Tp2.Agent1", null);
            ag2.start();
        } catch (StaleProxyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

after execution :

```
Hello world, My name is Agent2
Hello world, My name is Agent1
```

Two new agents have been inserted into the JADE platform.



3.5.Developing JADE Agents by Program java (Product Selling):

1/ Develop a Jade agent class, which receives data as arguments and displays it.

```java
package Tp2;
import jade.core.Agent;

public class Agdisplay extends Agent {

    float price;
    String name;
    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        super.setup();//pour lancer les traitements de l'agent
        Object[] args = getArguments();
        if(args!=null) {
            name = (String) args[0];
            price = Float.valueOf((String) args[1]).floatValue();
        }
        System.out.println("In Agent : "+getLocalName()+" Sell product "+name+" at price "+price);

    }

}
```

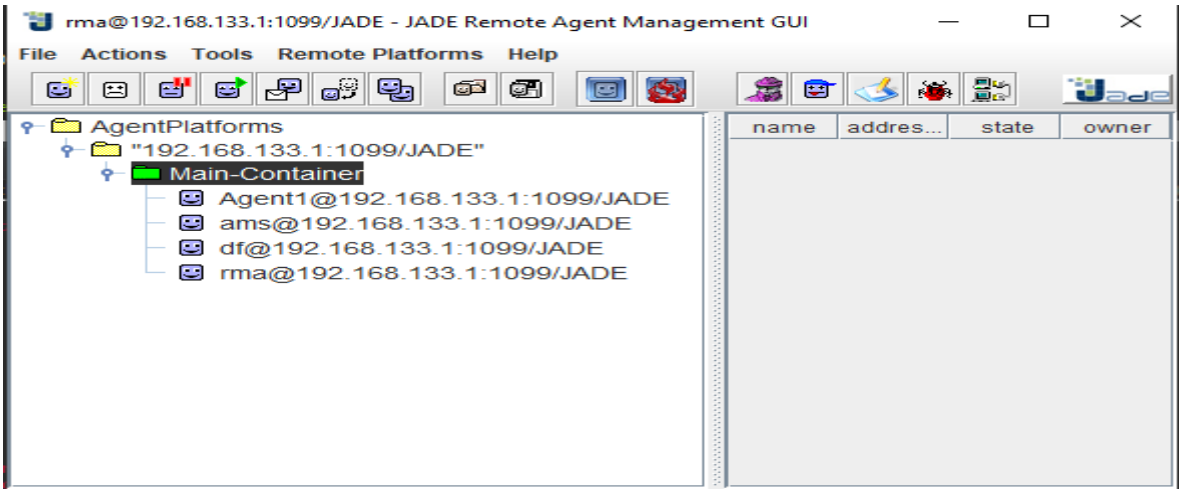
2/ Give the steps for launching an agent by program java:

```java
package Tp2;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;

public class Agent_launcher {

    public static void main(String[] args) {
        Object[] arg1 = {"Laptop","999.99"};
        //Object[] arg2 = {"keybord","255.50"};

        try {
            Runtime rt = Runtime.instance();
            ProfileImpl p = new ProfileImpl();
            p.setParameter(Profile.LOCAL_HOST,"localhost");
            p.setParameter(Profile.LOCAL_PORT,"1099");
            p.setParameter(Profile.GUI,"true");
            ContainerController mc = rt.createMainContainer(p);

            AgentController ag1 = mc.createNewAgent("Agent1", "Tp2.Agdisplay", arg1);
            ag1.start();

        //AgentController ag2 = mc.createNewAgent("buyer2", "Tp2.Agdisplay",arg2);
            //ag2.start();

        } catch (StaleProxyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

after execution :

```
-----------------------------------------
In Agent : Agent1 Sell product Laptop at price 999.99
```

A new agent has been inserted into the JADE platform.



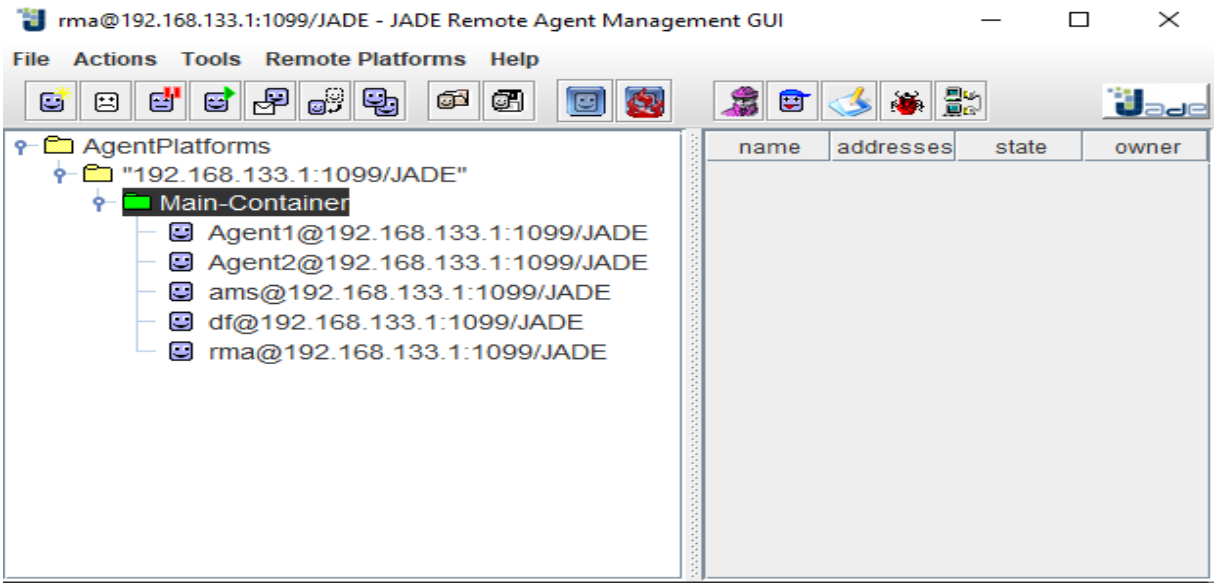
3/ Give the steps for launching two agents by program java

```java
package Tp2;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;

public class Agent_launcher {

    public static void main(String[] args) {
        Object[] arg1 = {"Laptop","999.99"};
        Object[] arg2 = {"keybord","255.50"};

        try {
            Runtime rt = Runtime.instance();
            ProfileImpl p = new ProfileImpl();
            p.setParameter(Profile.LOCAL_HOST,"localhost");
            p.setParameter(Profile.LOCAL_PORT,"1099");
            p.setParameter(Profile.GUI,"true");
            ContainerController mc = rt.createMainContainer(p);

            AgentController ag1 = mc.createNewAgent("Agent1", "Tp2.Agdisplay", arg1);
            ag1.start();

        AgentController ag2 = mc.createNewAgent("Agent2", "Tp2.Agdisplay",arg2);
            ag2.start();

        } catch (StaleProxyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

after execution :

```
In Agent : Agent2 Sell product keybord at price 255.5
In Agent : Agent1 Sell product Laptop at price 999.99
```

Two new agents have been inserted into the JADE platform.


rma@192.168.133.1:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
AgentPlatforms
"192.168.133.1:1099/JADE"
Main-Container
Agent1@192.168.133.1:1099/JADE
Agent2@192.168.133.1:1099/JADE
ams@192.168.133.1:1099/JADE
df@192.168.133.1:1099/JADE
rma@192.168.133.1:1099/JADE
name
addresses
state
owner

4. Chapter 3: JADE Communication

4.1. Introduction:

This chapter delves into communication within the JADE (Java Agent Development Framework) platform. Communication among agents is crucial in multi-agent systems, facilitating information exchange and action coordination. We explore various aspects of communication with JADE, from simple point-to-point messaging to topic-based broadcasting. Each practical exercise provides an opportunity to deepen understanding of communication mechanisms, strengthening the skills needed to develop robust and efficient multi-agent applications. By the end of the chapter, participants are better equipped to leverage JADE's communication capabilities in creating intelligent and scalable systems.

4.2. Point-to-point communication :

```java
package Pack_msg;

import jade.core.Agent;

public class A1_receiver extends Agent {
    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        addBehaviour(new CyclicBehaviour() {

            @Override
            public void action() {
                // TODO Auto-generated method stub
                ACLMessage msg = receive();
                MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM); //remplacer inform par propose
                //MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
            //  ACLMessage msg= receive(mt);
                if(msg != null) {
                    System.out.println("I am "+getLocalName()+" I received a message "
                    +msg.getContent()+" From the agent "+msg.getSender().getName());
                    ACLMessage reply = msg.createReply();
                    reply.setPerformative(ACLMessage.INFORM);
                    reply.setContent("Thank you");
                    send(reply);
                }
                block();

            }
        });
        super.setup();
    }

}
```

Explanation :

This JADE agent listens for incoming INFORM messages. Upon receiving one, it displays the message content and sends a polite "Thank you" response back to the sender using another INFORM message.

```java
package Pack_msg;

import jade.core.AID;

public class A2_sender extends Agent {
    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.setContent("Hello");
        msg.addReceiver(new AID("receiver",AID.ISLOCALNAME));
        send(msg);
        addBehaviour(new CyclicBehaviour() {

            @Override
            public void action() {
                // TODO Auto-generated method stub
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg = receive();
                if(msg != null) {
                    System.out.println("I am "+getLocalName()+" I received a reply "
                    +msg.getContent()+" From the agent "+msg.getSender().getName());
                }
                block();

            }
        });
        super.setup();
    }
}
```

Explanation :

This second JADE agent acts as the initiator. It constructs an INFORM message with the greeting "Hello" and sends it to a local agent identified by its name. The agent then listens for responses using a cyclic behavior, displaying the content of any

```java
package Pack_msg;

public class Agent_launcher {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String [] jadeArg = new String [2];
        StringBuffer SbAgent = new StringBuffer();
        SbAgent.append("sender:Pack_msg.A2_sender;");
        SbAgent.append("receiver:Pack_msg.A1_receiver");
        jadeArg[0] = "-gui";
        jadeArg[1] = SbAgent.toString();
        jade.Boot.main(jadeArg);

    }

}
```

Explanation :

- It is a class containing the **main()** method to launch the JADE platform with the previously defined agents.
- It creates a string array **jadeArg** to store the command-line arguments for launching JADE.
- It creates a string **SbAgent** containing the definition of the agents to be launched. In this example, it contains the class names of the agents with their full paths.
- It passes the **-gui** argument (to display the JADE graphical interface) and the agents' definition to the **main()** method of the **jade.Boot** class.

```java
package Pack_msg;

import jade.core.ProfileImpl;

public class SimpleContainer {

    public static void main(String[] args) {
        try {
            Runtime rt = Runtime.instance();
            ProfileImpl p = new ProfileImpl(false); //false pour dire que ce n'est pas le main container
            p.setParameter(ProfileImpl.MAIN_HOST,"localhost");
            AgentContainer container = rt.createAgentContainer(p);
            container.start();

        } catch (ControllerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

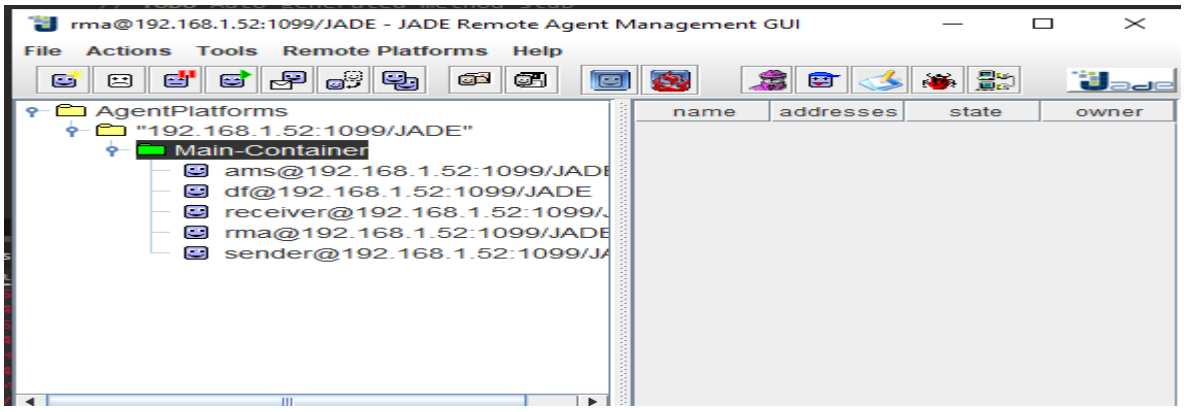
Explanation :

- It is a class that creates a simple JADE agent container.
- It uses the JADE API to create an instance of the runtime, define a profile for the container (specifying the main host), and then create and start an agent container.
- In this example, this container is used to run the **A1_receiver** and **A2_sender** agents.

After execution:

```
I am receiver I received a message Hello From the agent sender@192.168.1.52:1099/JADE
I am sender I received a reply Thank you From the agent receiver@192.168.1.52:1099/JADE
```

Two new agents have been inserted into the JADE platform.



4.3. Message filtering:

With "propose," it's the same thing, we just change "inform" to "propose":

This sentence means that the behavior of the agents remains similar, but instead of handling messages with the **INFORM** performatif, they now handle messages with the **PROPOSE** performatif.

Here's a brief explanation of how **PROPOSE** works and a demonstration:

In JADE, the **PROPOSE** performatif is typically used when an agent wants to suggest or propose something to another agent in a negotiation or decision-making scenario. For example, Agent A might propose a price to Agent B in a negotiation for a product.

To demonstrate this, let's modify the **A1_receiver** agent to handle messages with the **PROPOSE** performatif instead of **INFORM**. The other agents and setup remain the same.

```java
package Pack_msg;

import jade.core.Agent;

public class A1_receiver extends Agent {
    @Override
    protected void setup() {
        // TODO Auto-generated method stub
        addBehaviour(new CyclicBehaviour() {

            @Override
            public void action() {
                // TODO Auto-generated method stub
                //ACLMessage msg = receive();
                //MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM); //remplacer inform par propose
                MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
                ACLMessage msg= receive(mt);
                if(msg != null) {
                    System.out.println("I am "+getLocalName()+" I received a message "
                    +msg.getContent()+" From the agent "+msg.getSender().getName());
                    ACLMessage reply = msg.createReply();
                    reply.setPerformative(ACLMessage.INFORM);
                    reply.setContent("Thank you");
                    send(reply);
                }
                block();

            }
        });
        super.setup();
    }
}
```

4.4.Broadcast communication :

```java
package TP4;
import jade.core.Agent;


public class A1Topic extends Agent {
    protected void setup() {
        try {
            // Récupérer le gestionnaire de gestion des sujets
            TopicManagementHelper topicHelper = (TopicManagementHelper) getHelper(TopicManagementHelper.SERVICE_NAME);
            // Créer un sujet nommé "JADE"
            final AID topic = topicHelper.createTopic("JADE");
            // Enregistrement du sujet auprès du gestionnaire
            topicHelper.register(topic);

            addBehaviour(new CyclicBehaviour(this) {
                public void action() { // Méthode action qui sera exécutée de manière cyclique
                    ACLMessage msg = receive(MessageTemplate.MatchTopic(topic)); // Réception d'un message
                    if (msg != null) { // Vérification si un message a été reçu
                        System.out.println("AGENT " + getLocalName() + " Message about topic: '" + topic.getLocalName() +
                                "' received content is " + msg.getContent()); // Affichage du message reçu
                    } else {
                        block(); // Mise en attente de nouveaux messages
                    }
                }
            });
        } catch (ServiceException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Explanation :

- This class represents an agent that subscribes to a topic named "JADE" and listens for messages published on that topic.
- In the setup() method, it creates a topic named "JADE" using the TopicManagementHelper, registers the topic, and adds a CyclicBehaviour to continuously listen for messages on that topic.
- When a message is received, it checks if the message is related to the "JADE" topic and prints the content of the message.

```java
package TP4;

import jade.core.Agent;

public class A2Topic extends Agent {
    protected void setup() {
        try {
            // Récupérer le gestionnaire de gestion des sujets
            TopicManagementHelper topicHelper = (TopicManagementHelper) getHelper(TopicManagementHelper.SERVICE_NAME);
            // Créer un sujet nommé "JADE"
            final AID topic = topicHelper.createTopic("JADE");

            addBehaviour(new TickerBehaviour(this, 10) {
                protected void onTick() {
                    System.out.println("Agent " + getLocalName() + ": Envoi du message sur le sujet " + topic.getLocalName());
                    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                    // Ajouter le sujet en tant que destinataire du message
                    msg.addReceiver(topic);
                    // Définir le contenu du message comme le nombre d'intervalles de temps écoulés
                    msg.setContent(String.valueOf(getTickCount()));
                    // Envoyer le message
                    send(msg);
                }
            });
        } catch (ServiceException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Explanation :

- This class represents an agent that periodically sends messages to a topic named "JADE".
- In the setup() method, it creates a topic named "JADE" using the TopicManagementHelper.
- It then adds a TickerBehaviour to periodically send messages to the "JADE" topic.
- The content of the message is the number of tick intervals since the agent started.

```java
package TP4;

import jade.core.Profile;

public class SimpleContainer {

    public static void main(String[] args) {
        // Créer le profil par défaut avec l'interface graphique
        Profile profile = new ProfileImpl(true);
        profile.setParameter(Profile.GUI, "true");

        // Activer les services de gestion des sujets de message
        profile.setParameter(Profile.SERVICES, jade.core.messaging.TopicManagementService.class.getName());

        // Créer le conteneur d'agents
        AgentContainer container = jade.core.Runtime.instance().createMainContainer(profile);

        try {
            // Créer et lancer l'agent Sender
            AgentController senderController = container.createNewAgent("Ag1", "TP4.A2Topic", null);
            senderController.start();

            // Créer et lancer les agents Receiver
            for (int i = 2; i <= 4; i++) {
                AgentController receiverController = container.createNewAgent("Ag" + i, "TP4.A1Topic", null);
                receiverController.start();
            }
        } catch (StaleProxyException e) {
            e.printStackTrace();
        }
    }
}
```

Explanation :

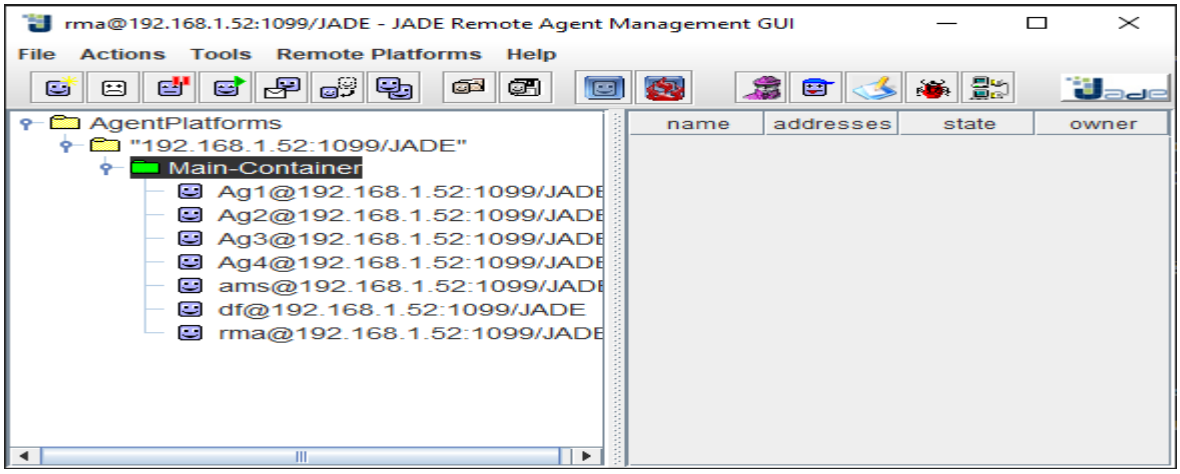
- This class is responsible for creating and launching agents within a JADE container.

- It creates a main container with a graphical interface and enables the messaging topic management service.
- It then creates and starts instances of agents A2Topic and A1Topic.

After execution:

```
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag2 Message about topic: 'JADE' received content is 1
AGENT Ag3 Message about topic: 'JADE' received content is 1
AGENT Ag4 Message about topic: 'JADE' received content is 1
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 2
AGENT Ag2 Message about topic: 'JADE' received content is 2
AGENT Ag4 Message about topic: 'JADE' received content is 2
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 3
AGENT Ag4 Message about topic: 'JADE' received content is 3
AGENT Ag2 Message about topic: 'JADE' received content is 3
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 4
AGENT Ag4 Message about topic: 'JADE' received content is 4
AGENT Ag2 Message about topic: 'JADE' received content is 4
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 5
AGENT Ag4 Message about topic: 'JADE' received content is 5
AGENT Ag2 Message about topic: 'JADE' received content is 5
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 6
AGENT Ag2 Message about topic: 'JADE' received content is 6
AGENT Ag4 Message about topic: 'JADE' received content is 6
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 7
AGENT Ag4 Message about topic: 'JADE' received content is 7
AGENT Ag2 Message about topic: 'JADE' received content is 7
Agent Ag1: Envoi du message sur le sujet JADE
AGENT Ag3 Message about topic: 'JADE' received content is 8
AGENT Ag4 Message about topic: 'JADE' received content is 8
AGENT Ag2 Message about topic: 'JADE' received content is 8
Agent Ag1: Envoi du message sur le sujet JADE
```

four new agents have been inserted into the JADE platform.



4.5. Serialization

```java
package Tp4;
import java.io.*;

import jade.util.leap.Serializable;
public class product implements Serializable {
    double price;
    String name;
    //seller agent

}
```

Explanation :

- This class defines a simple serializable data structure representing a product.
- It contains fields for the price and name of the product.

```java
package Tp4;

import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.core.Agent;

public class B1 extends Agent {
    protected void setup() {
        addBehaviour(new MyCyclicBehaviour());
    }

    private class MyCyclicBehaviour extends CyclicBehaviour {
        public void action() {
            ACLMessage m;
            product p;
            m = receive(); // Utilisation de la méthode receive() pour recevoir un message
            if (m != null) {
                try {
                    p = (product) m.getContentObject(); // Casting du contenu du message en objet de type Product
                    System.out.println("I am "+getLocalName()+" Received this product: " + p.name + " with the price: " + p.price);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Explanation :

- This class represents an agent that receives messages containing product information.
- It extends the Agent class and defines a CyclicBehaviour named MyCyclicBehaviour that listens for incoming messages.
- When a message is received, it attempts to extract the content as a product object and prints the product details.

```java
package Tp4;

import java.io.IOException;
import jade.lang.acl.ACLMessage;
import jade.core.AID;
import jade.core.Agent;

public class s2 extends Agent {
    product P = new product(); // Assurez-vous que la classe Product est correctement définie et importée
    ACLMessage m;

    protected void setup() {
        P.price=1000; // Utilisation de la méthode setPrice pour définir le prix du produit
        P.name="Keyboard"; // Utilisation de la méthode setName pour définir le nom du produit

        m = new ACLMessage(ACLMessage.INFORM);
        m.addReceiver(new AID("buyer", AID.ISLOCALNAME));
        try {
            m.setContentObject(P);
            m.setLanguage("JAVA");
            send(m); // Utilisation de la méthode send pour envoyer le message
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Explanation :

- This class represents an agent that sends a message containing a product object to another agent named "buyer".
- It initializes a product object with some sample data and sends it in an ACLMessage to the "buyer" agent.

```java
package Tp4;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;

public class Main {

    public static void main(String[] args) {
        // Création d'un profile pour la plateforme JADE
        Profile profile = new ProfileImpl();
        profile.setParameter(Profile.MAIN_HOST, "localhost"); // Spécifie l'hôte principal

        // Démarrage de la plateforme JADE
        Runtime runtime = Runtime.instance();
        try {
            AgentContainer container = runtime.createMainContainer(profile);

            // Démarrage de l'agent s2
            AgentController s2Controller = container.createNewAgent("s2", "Tp4.s2", null);
            s2Controller.start();

            // Démarrage de l'agent B1
            AgentController b1Controller = container.createNewAgent("b1", "Tp4.B1", null);
            b1Controller.start();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Explanation :Main

- This class serves as the entry point for launching the JADE platform and agents.
- It creates a main container and launches agents A2Topic and B1 within this container.

after execution

```
-----------------------------------------
Buyer received product: Keyboard with price: 1000.0
```

5. Chapter 4: JADE Behaviours

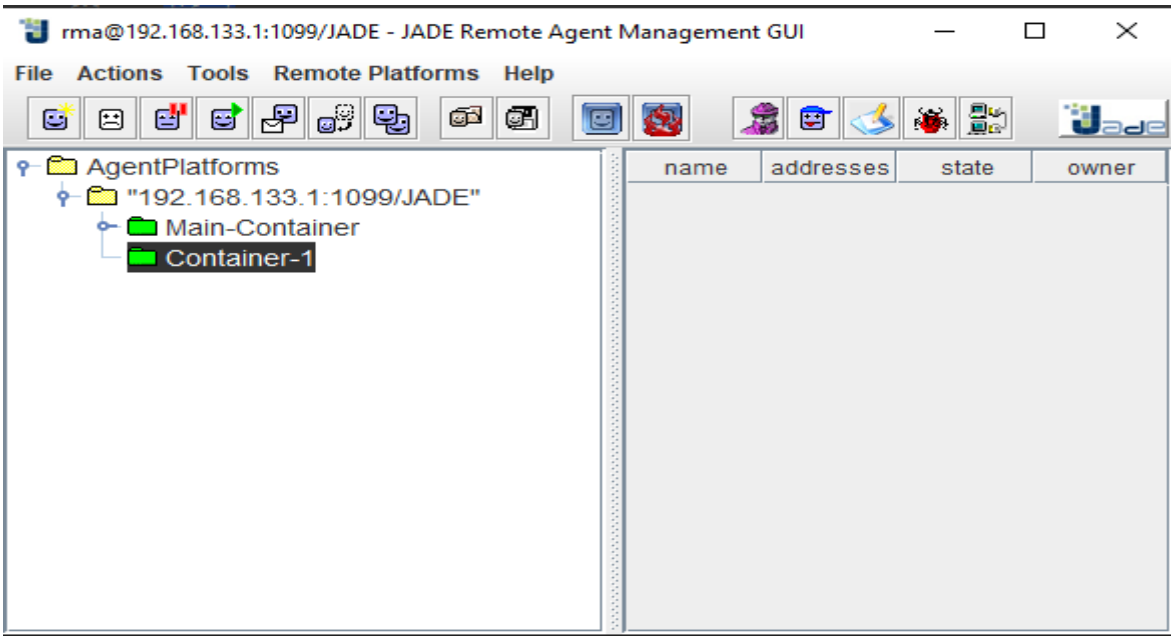
Exercice 1: (Agent migration : mobile agent)

```
package TP6;
import jade.core.Agent;
public class HelloAgent1 extends Agent {
    private String msg;
    protected void setup() {
        try {
        Object[] args = getArguments();
        if (args != null) {
            msg = (String) args[0];
            System.out.println("Hello world, I am an agent");
            System.out.println("My local name is " + getAID().getLocalName());
            System.out.println("My guid  is " + getAID().getName());
            System.out.println(msg);
        } else {
            System.out.println("No arguments provided.");
            doDelete();}

    } catch (Exception e) {
        e.printStackTrace();
    }
    }
    protected void takeDown() {
        System.out.println("End of the agent");
    }
    public void doMove(Location L) {
        System.out.println("Migration of the agent " + L.getName() );
        }
  protected void beforeMove() {
        System.out.println("Before migration of the agent " + this.getAID().getName());
        try {
            System.out.println("From container " + this.getContainerController().getContainerName());
        } catch (ControllerException e) {
            e.printStackTrace();    } }
   protected void afterMove() {
        System.out.println("After migration of the agent " + this.getAID().getName());
        try {
            System.out.println("To container " + this.getContainerController().getContainerName());
        } catch (ControllerException e) {
            e.printStackTrace();
        }}}
```

Classe SimpleContainer:

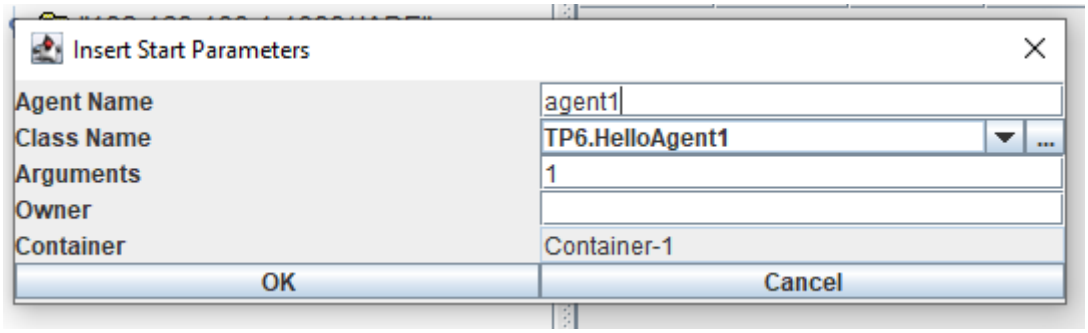
```
package TP6;
méta-heuristiques/src/metaProjectpartie02/Bso1.java
import java.awt.Container; // Importation de la classe Container de la bibliothèque AWT (Abstract Window Toolkit)

public class SimpleContainer { // Définition de la classe SimpleContainer

    public static void main(String[] args) { // Méthode main, point d'entrée du programme
        try { // Bloc try-catch pour la gestion des exceptions
            Runtime rt = Runtime.instance(); // Récupération de l'instance du runtime Jade
            ProfileImpl p = new ProfileImpl(); // Création d'un profil par défaut
            p.setParameter(Profile.MAIN_HOST, "localhost"); // Définition de l'hôte principal dans le profil

            AgentContainer container = rt.createAgentContainer(p); // Création d'un conteneur d'agents avec le profil spécifié
            container.start();

        } catch (Exception ex) { // Capture des exceptions et affichage des traces
            ex.printStackTrace(); // Affichage des traces de l'exception
        }
    }
}
```

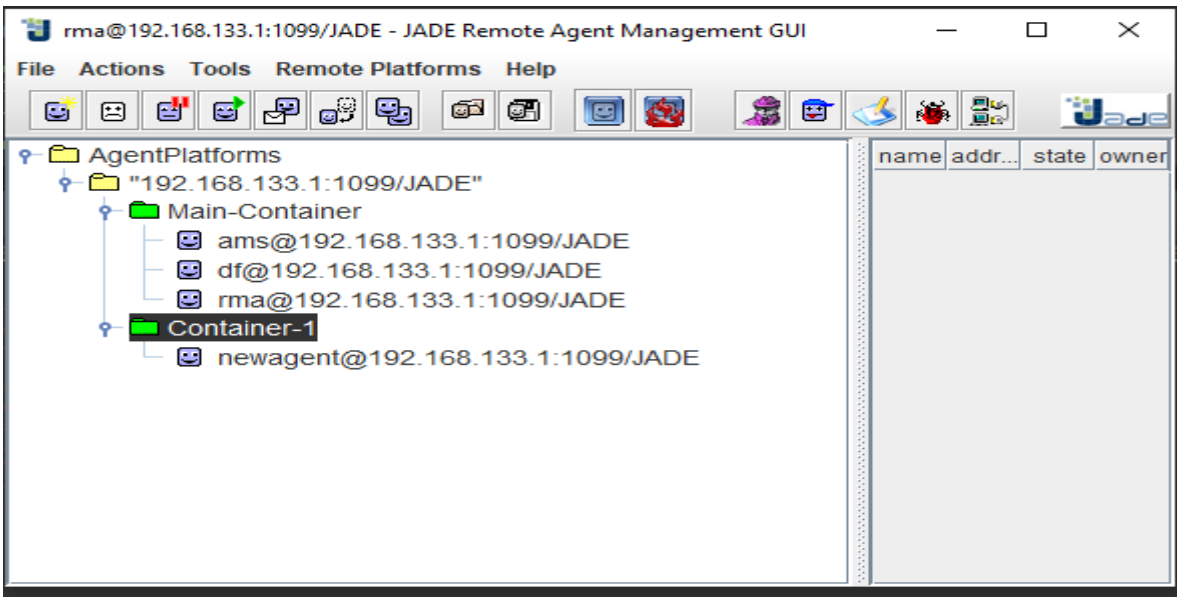
After exexution:

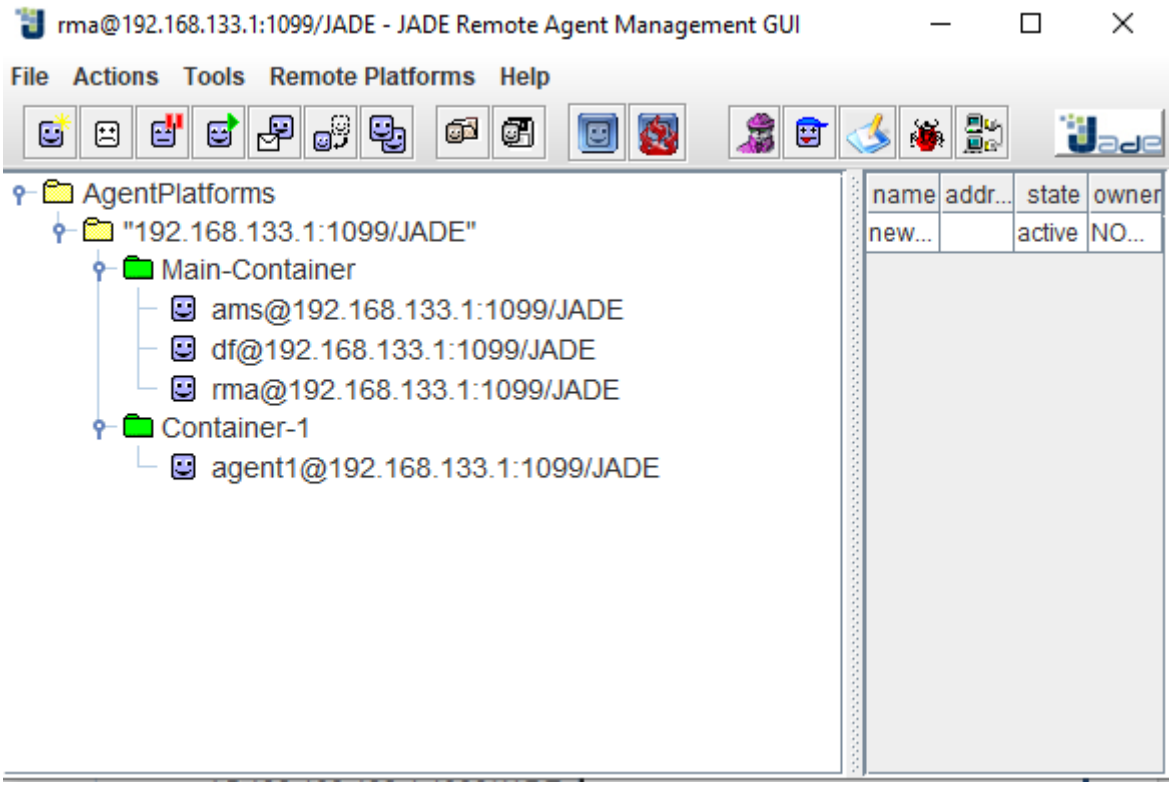


Explanation: new container is created container-1

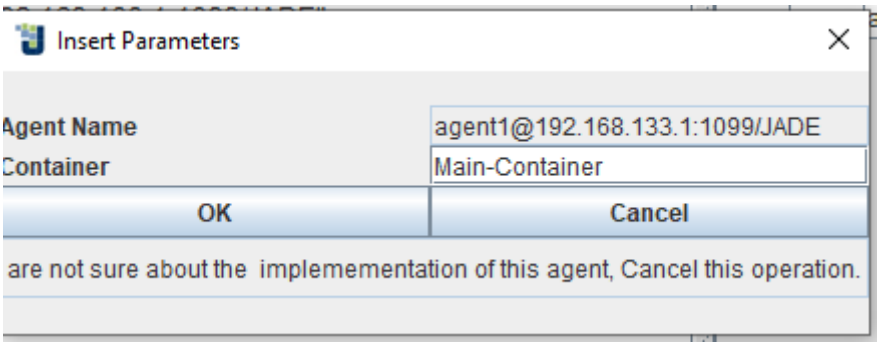
1. **Create new agent in Container-1:**



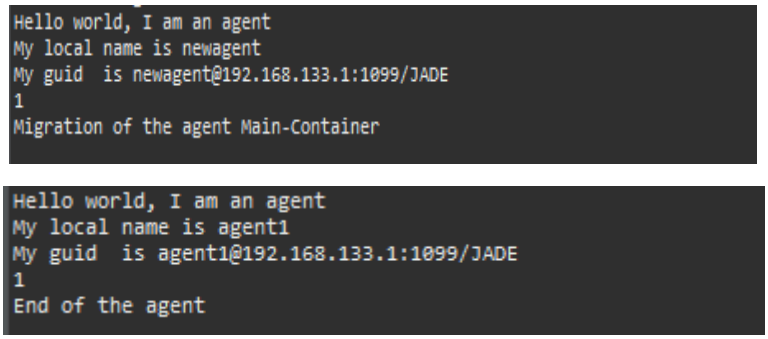




2/migration agent1 to main container



Result:



Explanation:

The provided Java code defines a helloAgent class representing an agent in the JADE platform. Here's a summary of the main functionalities of this class:

1. **Agent Initialization:**

Upon creation, the agent retrieves the passed arguments and displays a customized greeting message with its local name and the passed message as arguments.

2. **Termination Management:**

The agent has a takeDown() method that is called when the agent terminates, displaying a message indicating its end.

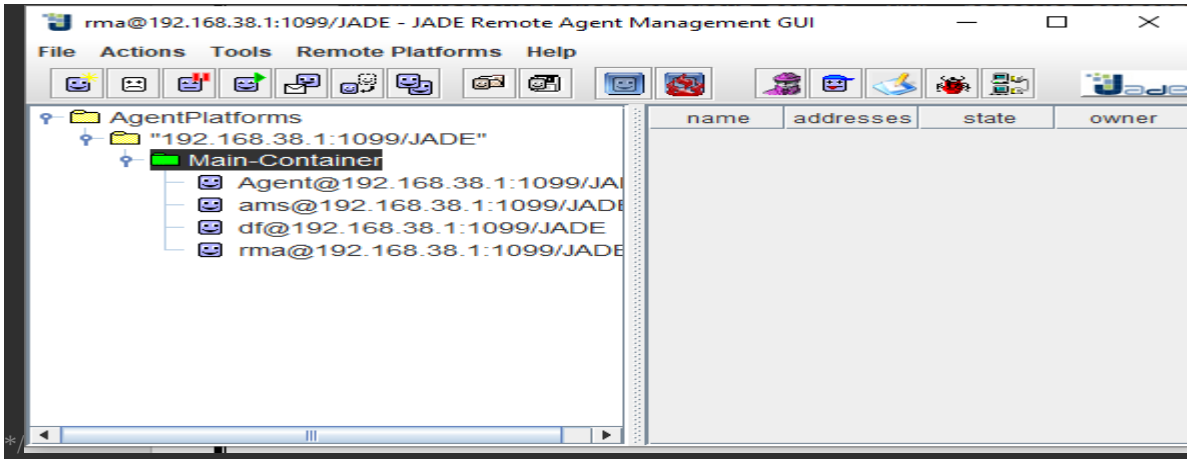
3. **Agent Migration:**

Methods (DOMove, beforeMove, afterMove) are provided to handle the migration of the agent to another location, but they are not currently implemented in the given code.

Exercise 2 : (Primitive behaviour)

```
package BehavPack;
import jade.core.Agent;
import jade.core.ServiceException;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.behaviours.WakerBehaviour;
import jade.core.AID;
import jade.core.messaging.TopicManagementHelper;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.ControllerException;
import jade.lang.acl.MessageTemplate;
public class AgentBehaviour extends Agent {
    protected void setup() {
        System.out.println("Sterting agent");
        addBehaviour(new Behaviour() {
            public void action() {
                System.out.println("this action is executed infinity of times");
            }

            @Override
            public boolean done() {
                // TODO Auto-generated method stub
                return false;
            }
        });
```

4. **Done(): Returns false, it executes infinitely.**




```
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
```

```java
package BehavPack;
import jade.core.Agent;
import jade.core.ServiceException;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.core.behaviours.WakerBehaviour;
import jade.core.AID;
import jade.core.messaging.TopicManagementHelper;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.ControllerException;
import jade.lang.acl.MessageTemplate;
public class AgentBehaviour extends Agent {
    protected void setup() {
        System.out.println("Sterting agent");
        addBehaviour(new Behaviour() {
            public void action() {
                System.out.println("this action is executed infinity of times");
            }

            @Override
            public boolean done() {
                // TODO Auto-generated method stub
                return true;
            }
        });
```

Done(): Returns true, it executes only once.

```
avr. 24, 2024 11:44:32 AM jade.core.messaging.M
INFO: MTP addresses:
http://DESKTOP-HBU9AIE:7778/acc
avr. 24, 2024 11:44:32 AM jade.core.AgentContai
INFO: ----------------------------------------
Agent container Main-Container@192.168.38.1 is
----------------------------------------------
Sterting agent
this action is executed infinity of times
```

1. **CyclicBehaviour:**

```java
    addBehaviour(new CyclicBehaviour(){

      public void action() {
          System.out.println("this action is executed.....times");
      }});
  /*
```

```
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
this action is executed infinity of times
```

2. **OneshotBehaviour:**

```
addBehaviour(new OneShotBehaviour() {
    public void action() {
        System.out.println("this action is executed just one times");
    }}

    );
```

```
Sterting agent
this action is executed just one times
```

3. **WakerBehaviour:**

```
addBehaviour(new WakerBehaviour(this,6000) {
    public void onWake() {
        System.out.println(getLocalName() + "I'm dying");
        doDelete();
        }}

    );
```

```
Sterting agent
AgentI'm dying
```

4. **TickerBehaviour :**

```
addBehaviour(new TickerBehaviour(this,30000) {

    @Override
    protected void onTick() {
        // TODO Auto-generated method stub
        System.out.println("this action is runned each 30 s");
    }});
```

```
Sterting agent
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
this action is runned each 30 s
```

Exercise 3 : (Composite behaviour)

```java
package Tp5;

import jade.core.Agent;

public class parrallel_behav extends Agent{
 protected void setup() {
  // TODO Auto-generated method stub
  System.out.println("I, Agent "+ getLocalName() + ", My adress is " + getAID());
  System.out.println("I execute several behaviors in parrallel");

  ParallelBehaviour paraB = new ParallelBehaviour();
  paraB.addSubBehaviour(langhelloBehaviour("عليكم السلام"));
  paraB.addSubBehaviour(langhelloBehaviour("bonjour"));
  paraB.addSubBehaviour(langhelloBehaviour("hello"));
  paraB.addSubBehaviour(langhelloBehaviour("buenos dias"));
  paraB.addSubBehaviour(langhelloBehaviour("salutation"));
  paraB.addSubBehaviour(langhelloBehaviour(""));
  addBehaviour(paraB);
  //super.setup();
 }
 private Behaviour langhelloBehaviour(String msg) {
  Behaviour b = new Behaviour(this) {
   int i = 0;

   @Override
   public void action() {
    // TODO Auto-generated method stub
    //System.out.println("Message : "+msg);
    System.out.printf("%s -> %s (%d/3)\n", new Object[] {getLocalName(), msg, (i+1)});
    i++;
   }

   @Override
   public boolean done() {
    // TODO Auto-generated method stub
    return i>=3;
   }


  };
  return b;
 }}
```

Main:

```java
package Tp5;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentContainer;
import jade.wrapper.AgentController;
import jade.wrapper.StaleProxyException;

public class Main {
    public static void main(String[] args) {
        Runtime rt = Runtime.instance();
        Profile p = new ProfileImpl();
        p.setParameter(Profile.MAIN_HOST, "localhost");
        p.setParameter(Profile.MAIN_PORT, "1099");
        p.setParameter(Profile.CONTAINER_NAME, "Main-Container");
        AgentContainer container = rt.createMainContainer(p);

        try {
            AgentController agent1 = container.createNewAgent("Agent1", "Tp5.parrallel_behav", null);
            AgentController agent2 = container.createNewAgent("Agent2", "Tp5.parrallel_behav", null);

            agent1.start();
            agent2.start();
        } catch (StaleProxyException e) {
            e.printStackTrace();
        }
    }
}
```

After execution :

```
I, Agent Agent1, My adress is ( agent-identifier :name Agent1@192.168.133.1:1099/JADE  :addresses (sequence http://DESKTOP-QBKJI11:7778/acc ))
I execute several behaviors in parrallel
I, Agent Agent2, My adress is ( agent-identifier :name Agent2@192.168.133.1:1099/JADE  :addresses (sequence http://DESKTOP-QBKJI11:7778/acc ))
I execute several behaviors in parrallel
Agent2 -> عليكم السلام (1/3)
Agent1 -> عليكم السلام (1/3)
Agent2 -> bonjour (1/3)
Agent1 -> bonjour (1/3)
Agent2 -> hello (1/3)
Agent1 -> hello (1/3)
Agent2 -> buenos dias (1/3)
Agent1 -> buenos dias (1/3)
Agent2 -> salutation (1/3)
Agent1 -> salutation (1/3)
Agent2 ->  (1/3)
Agent1 ->  (1/3)
Agent2 -> عليكم السلام (2/3)
Agent1 -> عليكم السلام (2/3)
Agent1 -> bonjour (2/3)
Agent2 -> bonjour (2/3)
Agent1 -> hello (2/3)
Agent2 -> hello (2/3)
Agent1 -> buenos dias (2/3)
Agent2 -> buenos dias (2/3)
Agent1 -> salutation (2/3)
Agent2 -> salutation (2/3)
Agent1 ->  (2/3)
Agent1 -> عليكم السلام (3/3)
Agent2 ->  (2/3)
Agent2 -> عليكم السلام (3/3)
Agent1 -> hello (3/3)
Agent2 -> hello (3/3)
Agent1 -> salutation (3/3)
Agent2 -> salutation (3/3)
Agent1 -> bonjour (3/3)
```

```
Agent2 -> bonjour (3/3)
Agent1 ->  (3/3)
Agent2 ->  (3/3)
Agent1 -> buenos dias (3/3)
Agent2 -> buenos dias (3/3)
```

Explanation:

This Java code creates an agent using JADE that executes multiple behaviors concurrently. The agent displays its name and address upon initialization, then creates a ParallelBehaviour to run several instances of the langhelloBehaviour behavior, each displaying a specific message multiple times. Once the behaviors are added, the agent executes them in parallel

until each has been executed three times. In summary, the agent performs tasks simultaneously while maintaining coherent behavior in its multi-agent environment.

6. Conclusion :

The report provides an in-depth exploration of expert systems and the JADE platform, offering readers a solid understanding of the foundations of artificial intelligence and software agent development. By combining theory with practical examples, it illuminates the core principles of AI while providing valuable insights into the real-world applications of these technologies. Through a detailed analysis of both forward and backward chaining methods, as well as communication mechanisms within JADE, the report equips readers with essential knowledge and skills to tackle the challenges of AI and multi-agent systems. In conclusion, it serves as a valuable resource for researchers, developers, and AI enthusiasts, offering a comprehensive and practical overview of recent advances in these fields.