Lab Reports

Course: Agent Technology

Instructor: Mrs. B. Dellal-Hedjazi

Submitted by: SAYOUD Maissa

ID: 191931040670

Dated: May 25th, 2024

Department of Artificial Intelligence and Data Science

Masters in Intelligent Computer Systems

Faculty of Computer Science

USTHB

Table of content

Lab Report #01

Lab#01: Forward Chaining and Backward Chaining

Introduction

In this lab, we implemented the backward chaining and forward chaining algorithms in Java. Additionally, we integrated a graphical user interface (GUI) for a better visualization.

Note: We have already presented the Forward Chaining Algorithm in a google meet session.

1. Forward chaining

Artificial intelligence and expert systems employ the forward chaining technique to draw inferences from a starting collection of facts. In this approach, the system begins with the known facts and works its way up to the desired outcome by repeatedly applying inference rules to infer new data.

1.1. Basic Steps of the Forward Chaining Algorithm

1. Initialization:

We start with an initial set of known facts which is our knowledge base and we define after that a set of rules.

2. Rule Application:

In the loop, for each rule, we check if its premises are satisfied by the current set of known facts. If the premises are satisfied, we apply the rule to get the conclusions.

3. Updating Knowledge Base:

We add the conclusions derived from each rule to the knowledge base.

4. Goal Checking:

We Repeat the process of rule application and knowledge base update until we reach our goal or no other deductions can be applied. If the goal is reached, this is the end of the algorithm with success. If the goal is not reached yet, it continues until no more rules can be applied.

5. In the end, we indicate whether the goal is achievable based on the available knowledge and rules.

1.2. Implementation Details

In this section, we detail the implementation of each class involved in the Java code:

Main Class: It displays the graphical user interface (GUI), sets its components up, initializes the knowledge base rules and handles the interaction between the GUI and the forward chaining algorithm.

Rule Class: The Rule class represents individual rules in the knowledge base. Each rule consists of premises (P) and a conclusion (C). This class stores them of a rule, keeps track of the state of the rule.

ChooseRule Class: It selects a rule from a list of candidate rules.

ForwardChaining Class: It simply implements the forward chaining algorithm. It returns a boolean value for whether the goal has been proven or not.

1.3. Example Scenario

We have this starting Rule base where all of the rules are previously applicable:

Rule 1: Premises: A B - Conclusion: F

Rule 2: Premises: F H - Conclusion: I

Rule 3: Premises: D H G - Conclusion: A

Rule 4: Premises: O G - Conclusion: H

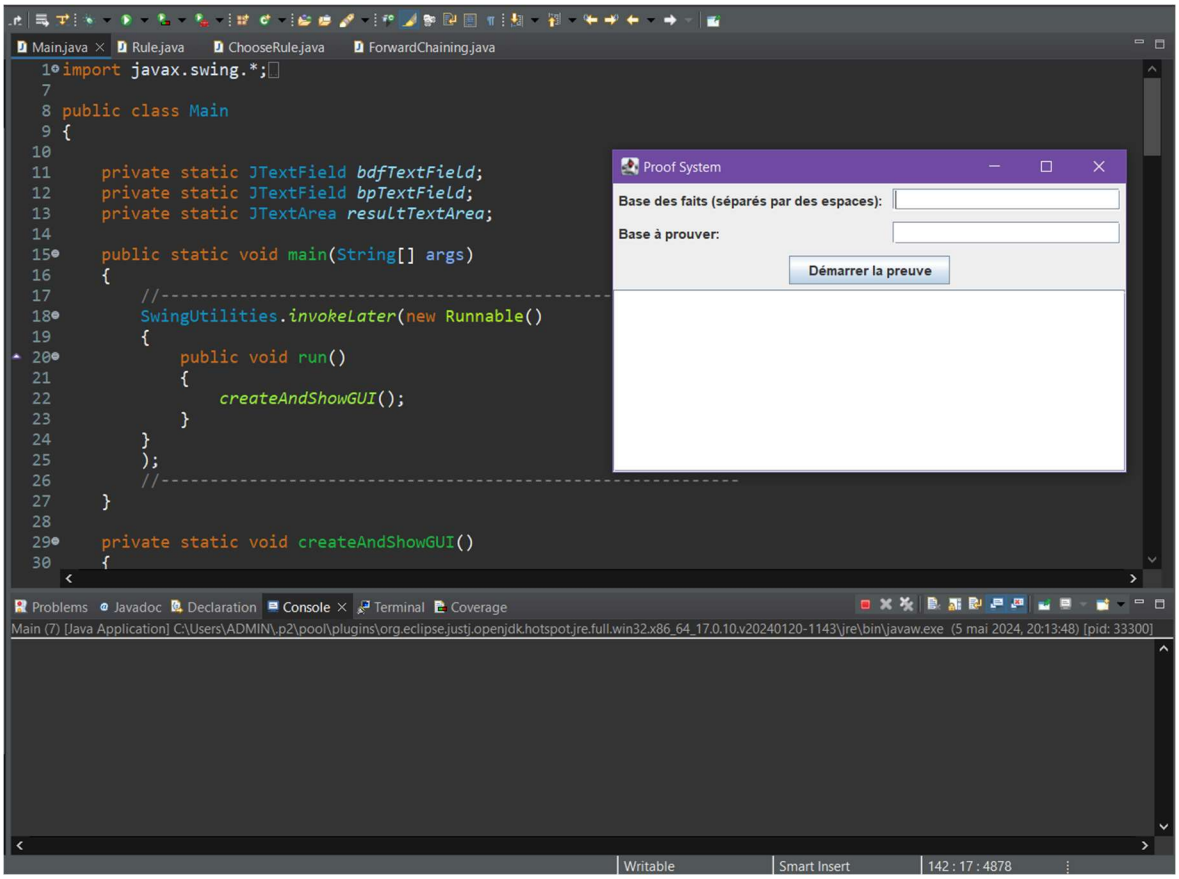
Rule 5: Premises: E H - Conclusion: B

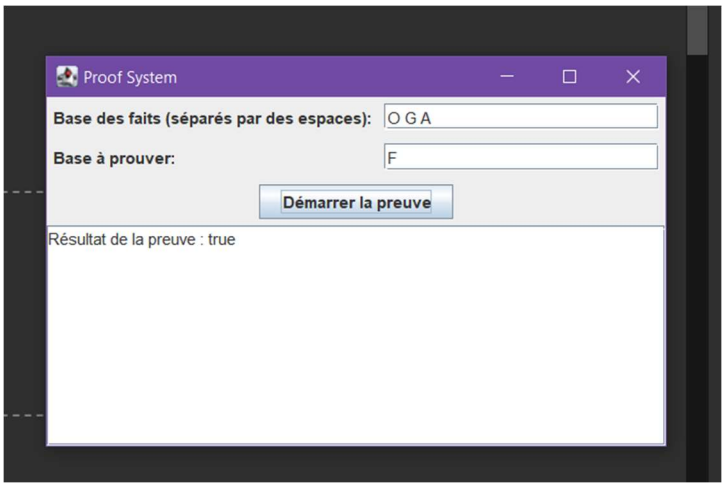
Rule 6: Premises: G A - Conclusion: B

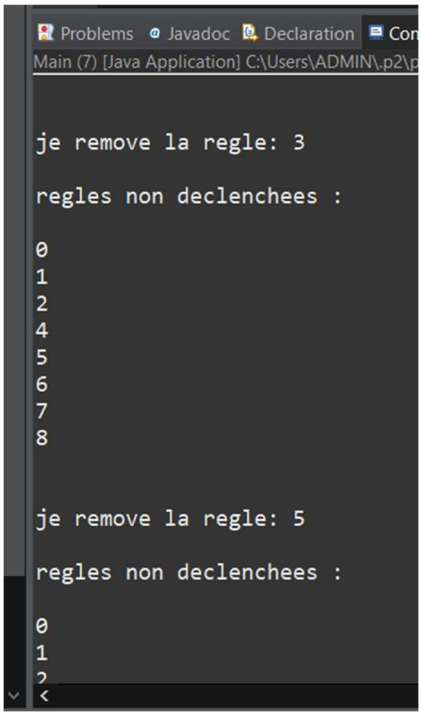
Rule 7: Premises: G H - Conclusion: P

Rule 8: Premises: G H - Conclusion: O

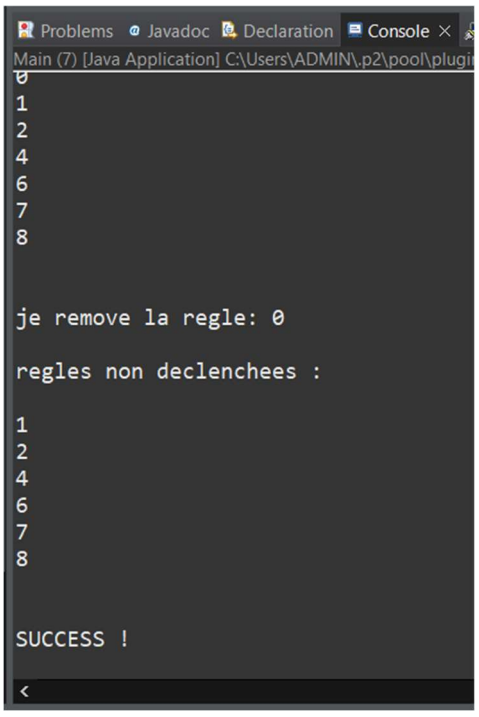
Rule 9: Premises: D O G - Conclusion: J


Main.java
Rule.java
ChooseRule.java
ForwardChaining.java
import javax.swing.*;
public class Main
{
private static JTextField bdfTextField;
private static JTextField bpTextField;
private static JTextArea resultTextArea;
public static void main(String[] args)
{
//----------
SwingUtilities.invokeLater(new Runnable()
{
public void run()
{
createAndShowGUI();
}
}
);
//----------
}
private static void createAndShowGUI()
{
Proof System
Base des faits (séparés par des espaces):
Base à prouver:
Démarrer la preuve
Problems
Javadoc
Declaration
Console
Terminal
Coverage
Main (7) [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (5 mai 2024, 20:13:48) [pid: 33300]
Writable
Smart Insert
142 : 17 : 4878


Proof System
Base des faits (séparés par des espaces):
O G A
Base à prouver:
F
Démarrer la preuve
Résultat de la preuve : true



(suite)



2. Backward Chaining

Another technique for reasoning in artificial intelligence and expert systems is backward chaining, which simply starts with the goal and moves backward to the known facts in order to demonstrate the given goal.

2.1. Basic Steps of the Backward Chaining Algorithm

The steps of backward chaining were already explained and discussed in the lab session, so we recall the algorithm:

```
Fonction chaînageArrière
Paramètres : in BR, in BF, in listeButs.
  if estVide(listeButs) then
    res ← SUCCES
  else
    if demBut(premier(listeButs)) then
      res ← chaînageArriere(suite(listeButs))
    else
      res ← ECHEC
    end if
  end if
  retourner res
```

```
Fonction demBut
Paramètres : in BR, in BF, in but.
  if but ∈ BF then
    res ← SUCCES
  else
    regles ← BR; res ← ECHEC
    while regles ≠ ∅ et res ≠ SUCCES do
      r ← choix(regles); regles ← regles - {r}
      if But ∈ conclusion(r) then
        res ← chaînageArrière(BR, BF, premisse(r))
      end if
    end while
    retourner res
  end if
```

2.2. Implementation Details

Even though the backward chaining approach was implemented during the lab session, I additionally integrated a graphical user interface (GUI) in the backward chaining too.

Main Class: It sets up the graphical user interface (GUI), takes input for the base of facts (bdf) and the goal to prove (bp).

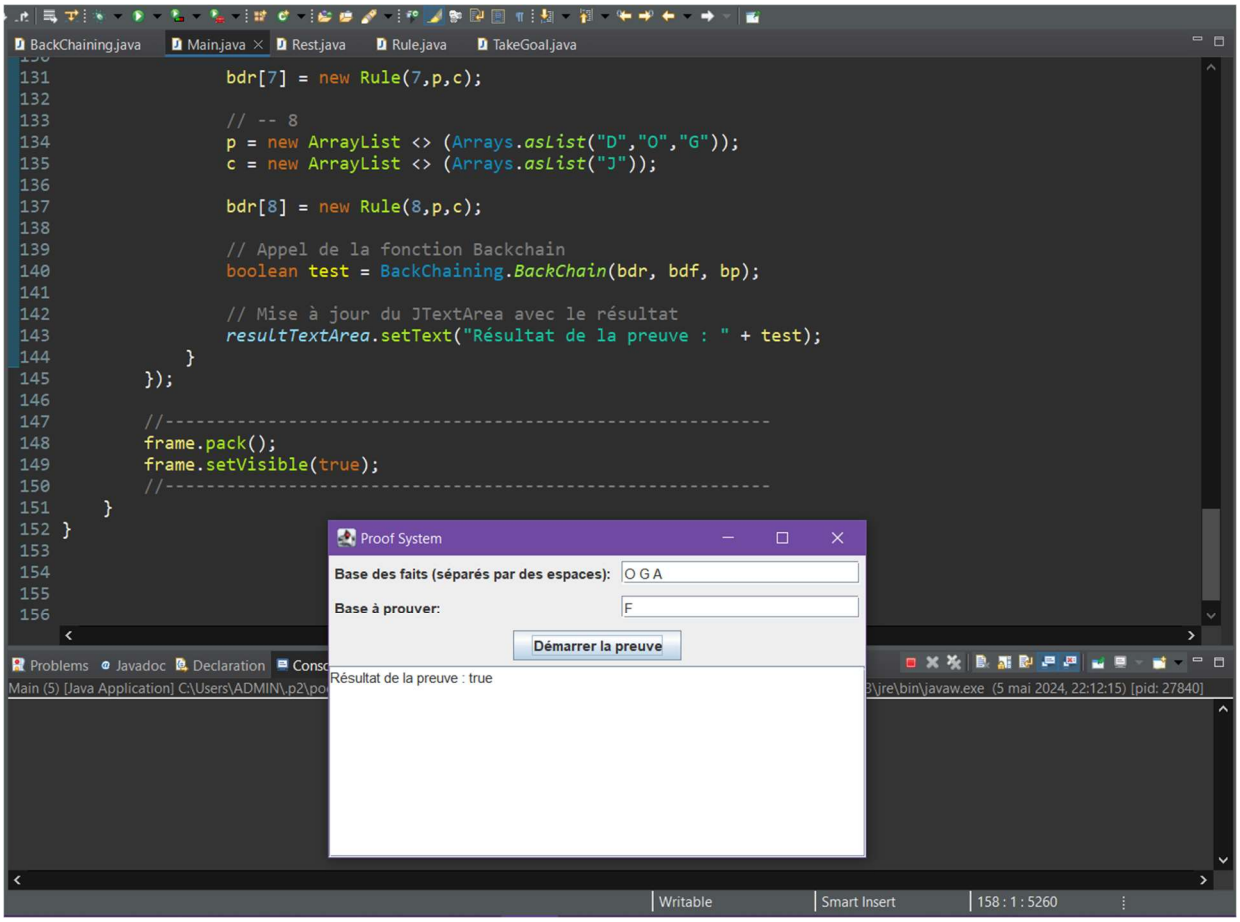
Backward Chaining: It contains the Back Chaining algorithm implementation. We recursively apply the backward chaining to try to prove the goal, starting from this latter to finding the base facts.

Rest Class: It excludes the first element of the given list and return the rest of it.

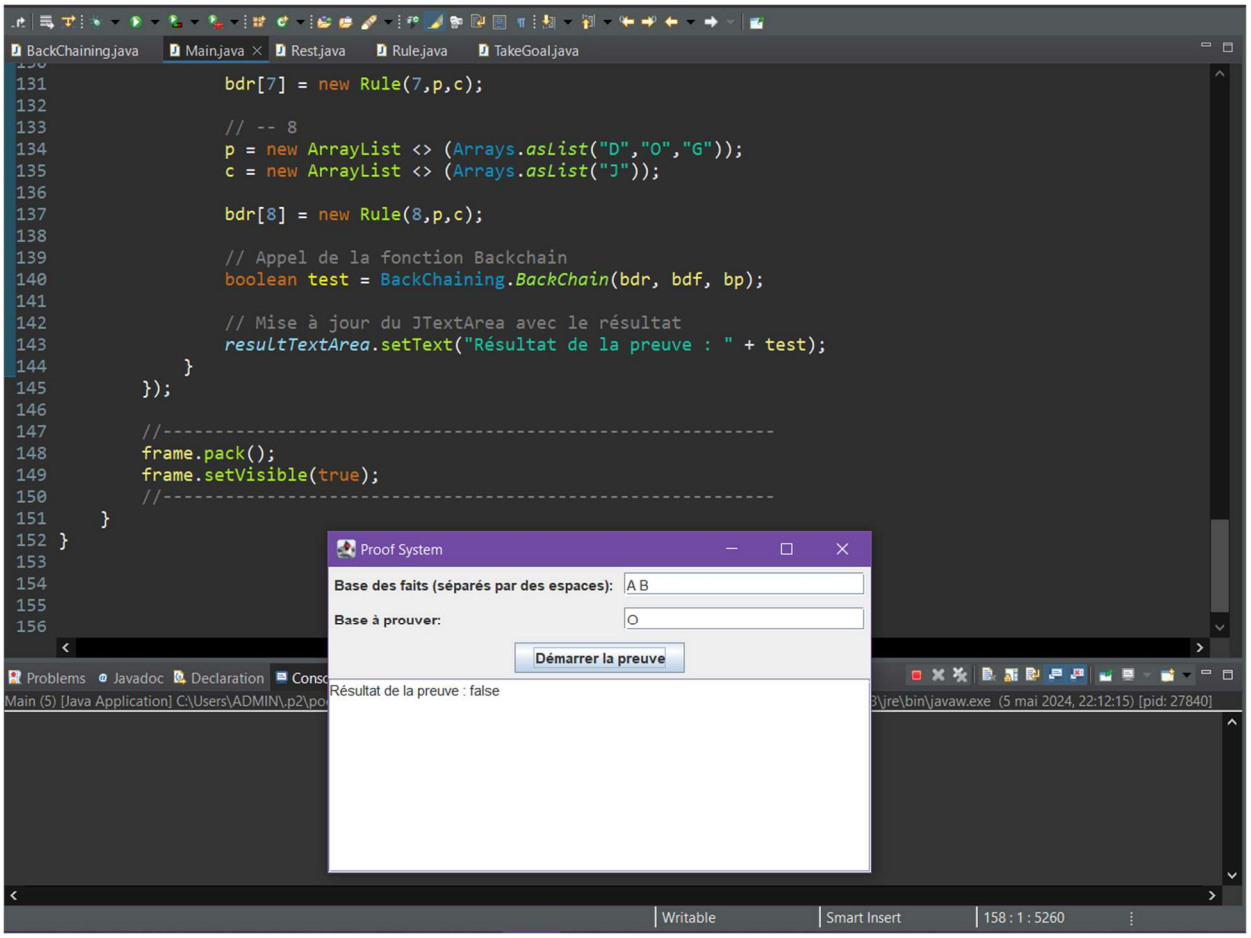
TakeGoal Class: It takes a goal and determines if this goal can be derived from the knowledge base, it finds also the conflict set (RA) which is consisted of rules with the goal as a conclusion and applies backward chaining to each rule in the conflict set.

2.3. Example Scenario

We use the same rules shown earlier with the forward chaining example, same collection of starting facts and the same goal.



Another example:



Conclusion

In this lab, we implemented both forward chaining and backward chaining algorithms in Java, we gained a deeper understanding of how they operate and their respective strengths and weaknesses.

Lab Report #02

Lab#02: Jade Introduction

Introduction

JADE provides a comprehensive platform for developing multi-agent systems (MAS) in Java.

In this lab, we explore various aspects of it through many exercises.

Please find the attached codes of the exercises in the **TechAgent_TP3_Agents.zip** file.

Exercise 01

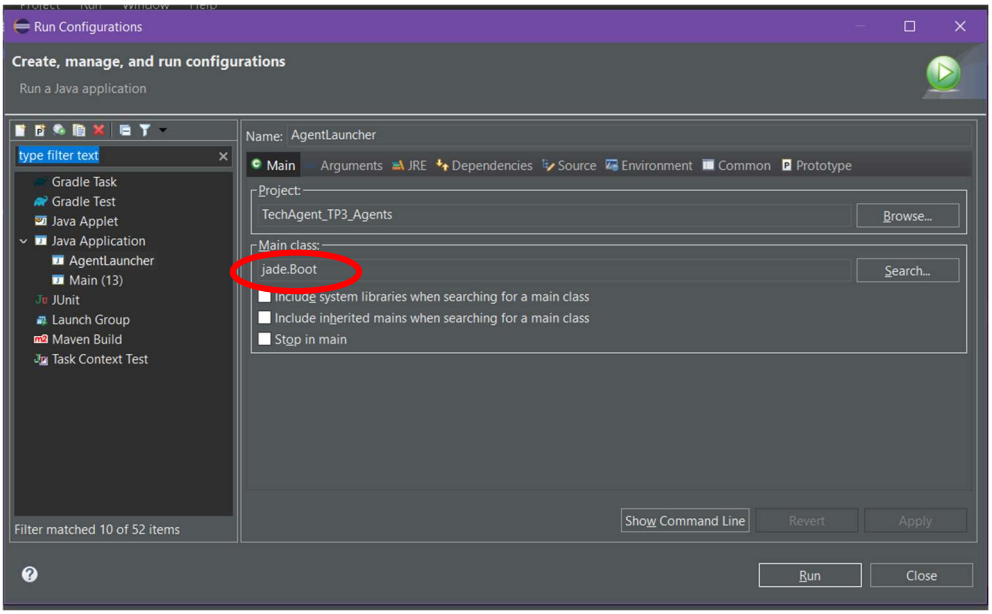
1/ Create an agent in JADE platform that displays the message « Hello world » followed by its name.

AgDisplay.java | AgDisplayA.java | AgentLauncher.java

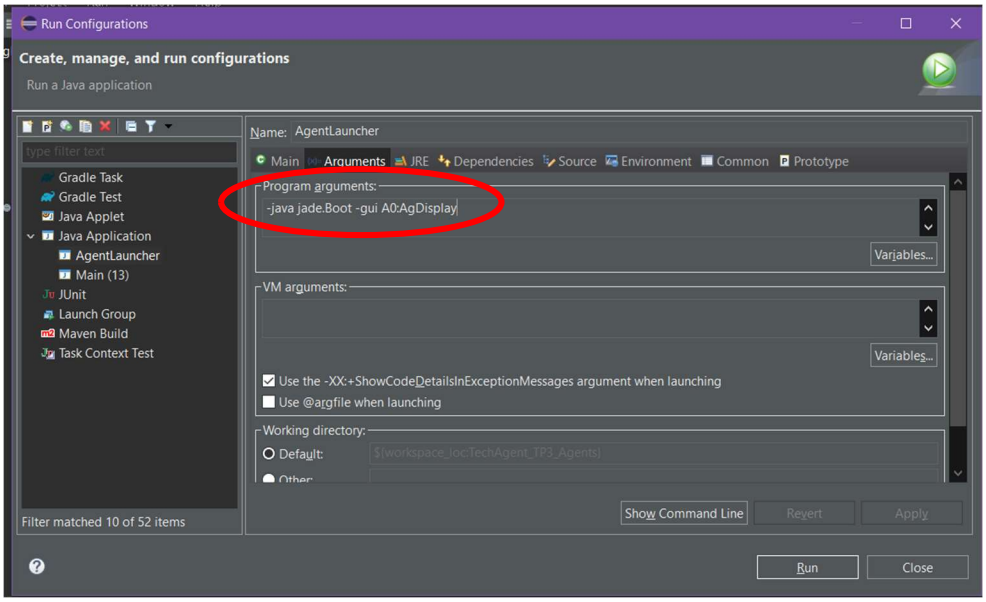
```
1  import jade.core.Agent;
2
3  public class AgDisplay extends Agent
4  {
5
6      private static final long serialVersionUID = 1L;
7
8      @Override
9      protected void setup()
10     {
11         System.out.println("\n\nMy name is : "+getLocalName());
12     }
13
14
15 }
16
```

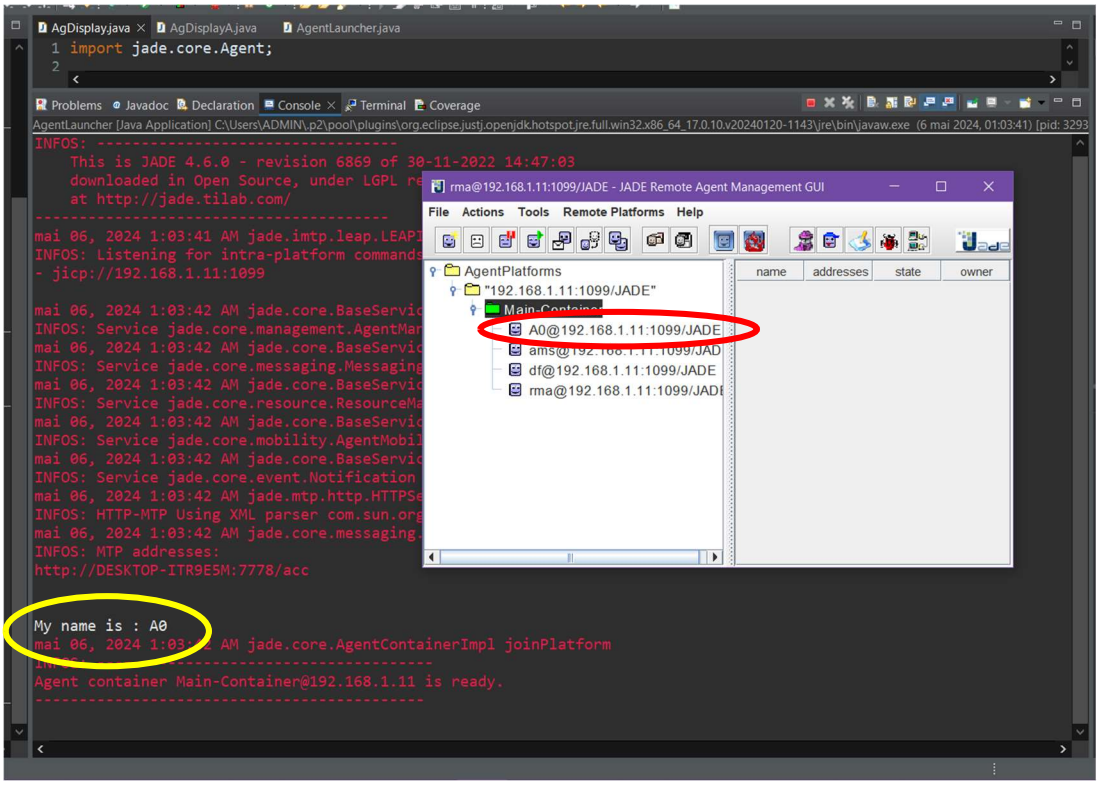
2/ Give the steps for launching one agent.

To launch one agent in JADE, we choose the **run as run configurations** option.



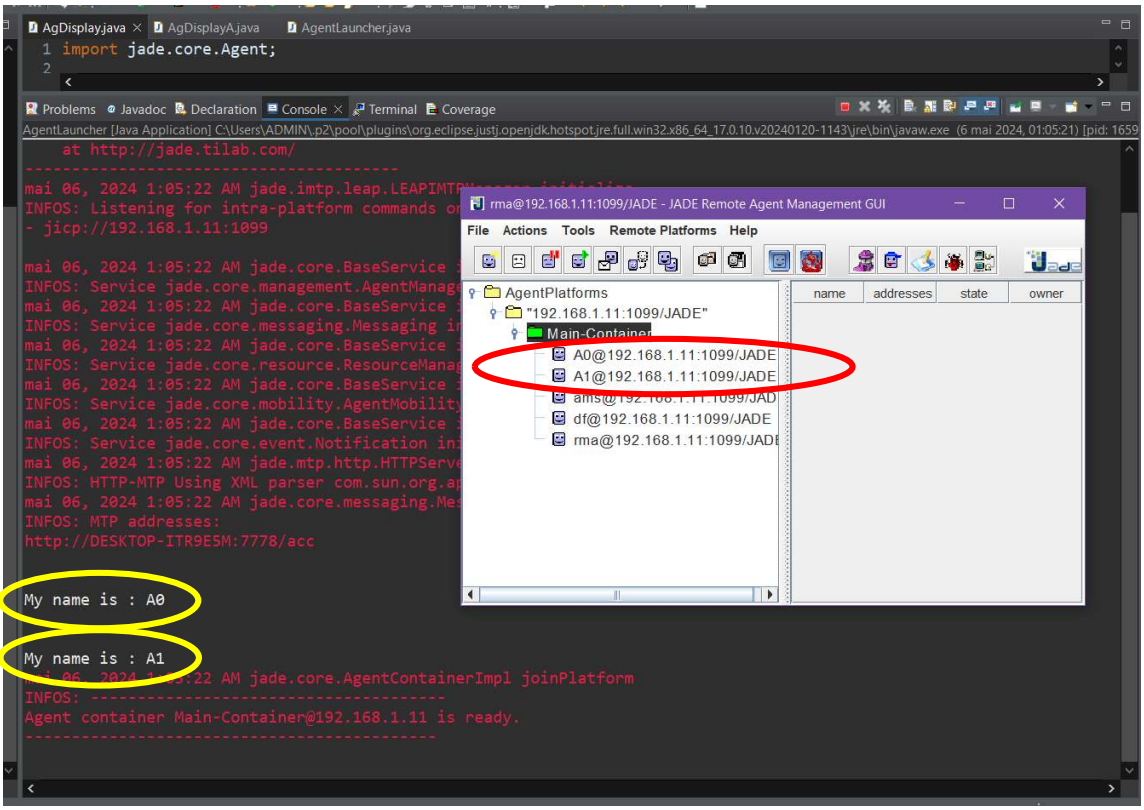
We specify the jade.Boot in the Main class field and then we type the following command in Arguments/Program arguments : **-java jade.Boot -gui A0:AgDisplay** in which we give a name to the agent and specify clearly the the class containing it.





3/ Give the steps for launching two agents.

To launch two agents we just change the command in Program Arguments to: **-java jade.Boot -gui A0:AgDisplay;A1:AgDisplay**



Exercise 02

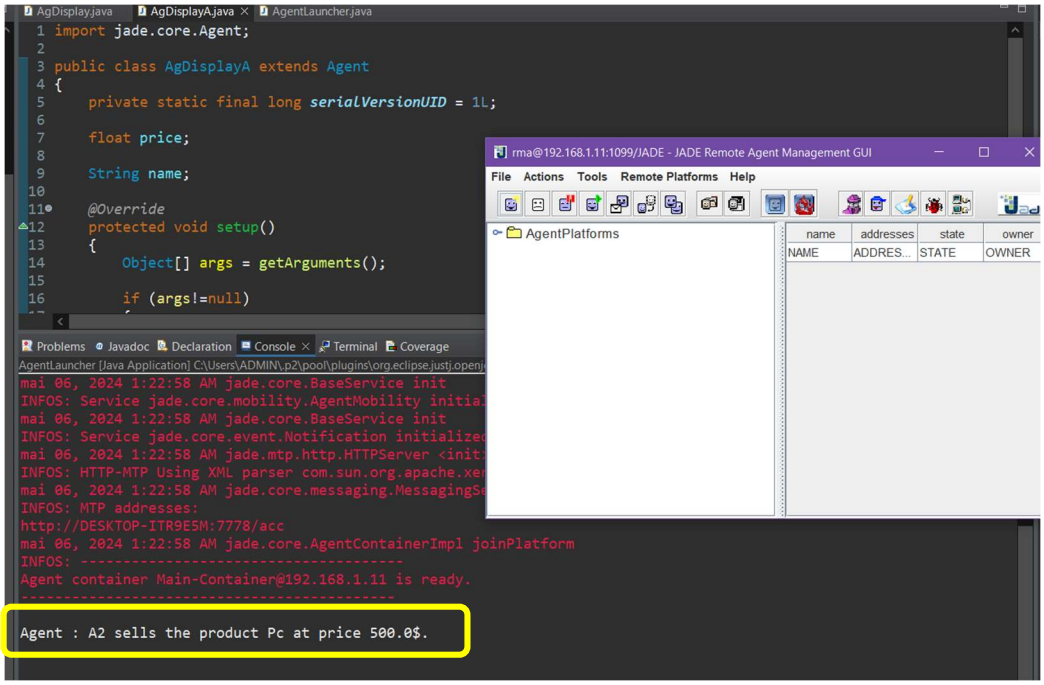
1/ Develop a Jade agent class, which receives data as arguments and displays it.

```java
import jade.core.Agent;

public class AgDisplayA extends Agent
{
    private static final long serialVersionUID = 1L;

    float price;

    String name;

    @Override
    protected void setup()
    {
        Object[] args = getArguments();

        if (args!=null)
        {
            name = (String)args[0];

            price = Float.valueOf((String)(args[1])).floatValue();

            System.out.println("\nAgent : "+getLocalName()+" sells the product "+name+" at price "+price+"$.\n\r

            doDelete(); //KILL agent ! dans ce cas "AP_DELETED" est à TRUE !
        }

    }
}
```

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

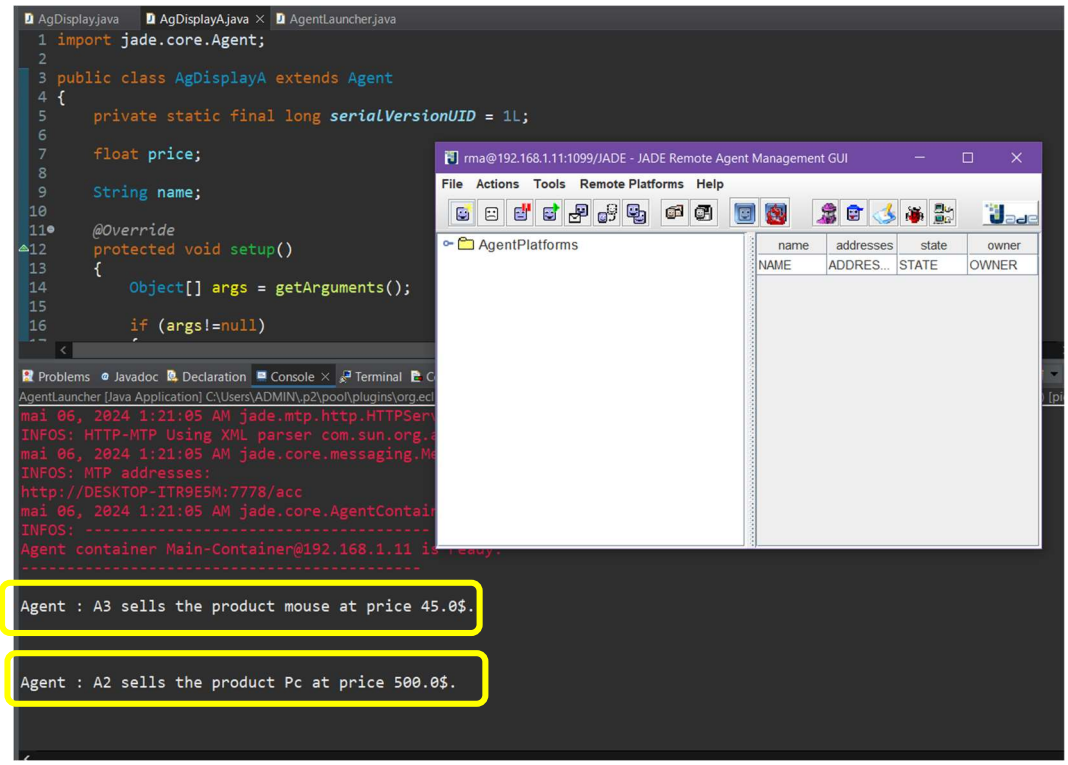
We follow the exact same steps as in the second question of the first exercise, with a slight change in the command:

-java jade.Boot -gui A2:AgDisplayA(Pc,500)



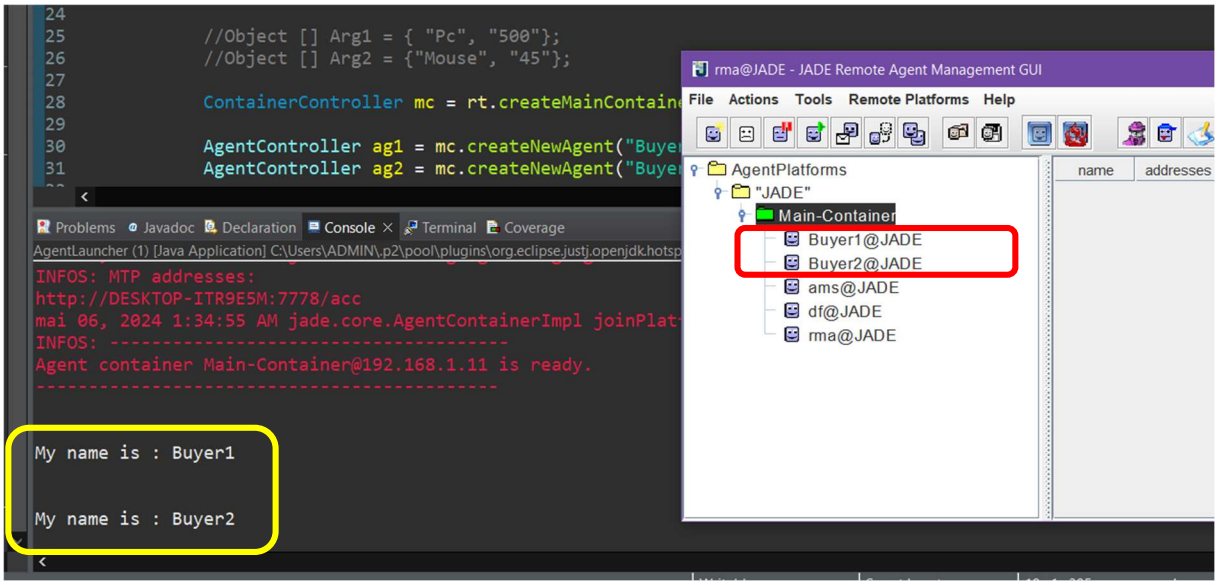
3/ Give the steps for launching two agents with passing arguments.

-java jade.Boot -gui A2:AgDisplayA(Pc,500);A3:AgDisplayA(mouse, 45)



Exercise 03 1/ Repeat exercise 1 to launch the agent by program. (Java)

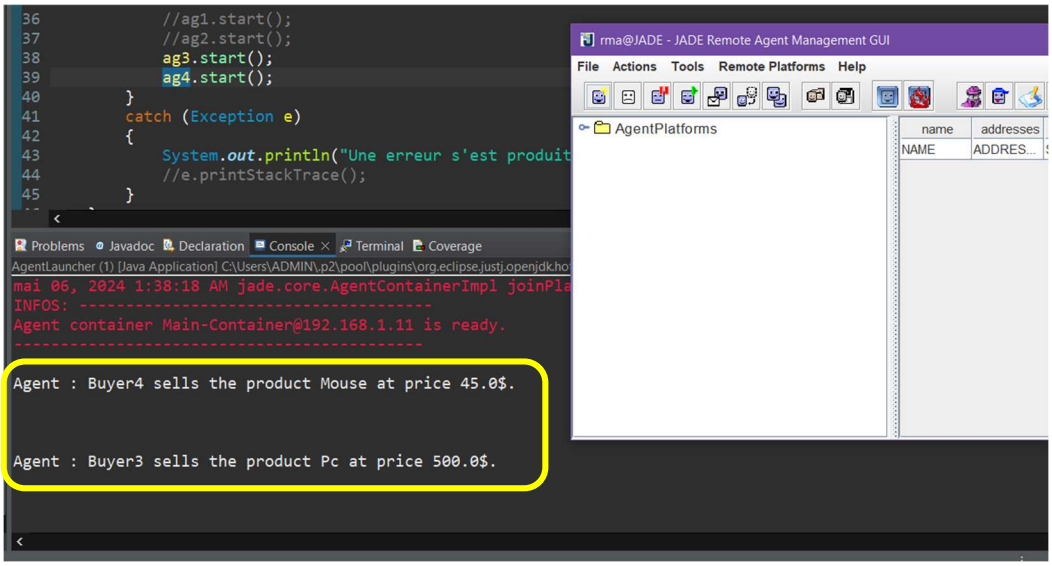
```java
public static void main(String[] args)
{
    try
    {
        Runtime rt = Runtime.instance();

        ProfileImpl p = new ProfileImpl("localhost",1099,"JADE");

        //pour afficher l'interface graphique il faudra rajouter les param:
        p.setParameter(Profile.LOCAL_HOST, "localhost");
        p.setParameter(Profile.LOCAL_PORT, "1099");
        p.setParameter(Profile.GUI, "true");

        //Object [] Arg1 = { "Pc", "500"};
        //Object [] Arg2 = {"Mouse", "45"};

        ContainerController mc = rt.createMainContainer(p);

        AgentController ag1 = mc.createNewAgent("Buyer1","AgDisplayA",null);
        AgentController ag2 = mc.createNewAgent("Buyer2","AgDisplayA",null);

        //AgentController ag3 = mc.createNewAgent("Buyer3","AgDisplayA",Arg1);
        //AgentController ag4 = mc.createNewAgent("Buyer4","AgDisplayA",Arg2);

        ag1.start();
        ag2.start();
        //ag3.start();
        //ag4.start();
    }
    catch (Exception e)
    {
        System.out.println("Une erreur s'est produite : " + e.getMessage());
        //e.printStackTrace();
    }
}
}
```



Exercise 04

1/ Repeat exercise 2 to launch the agent programmatically. Launch two agents, each with different input data (arguments).

```
12    public static void main(String[] args)
13    {
14        try
15        {
16            Runtime rt = Runtime.instance();
17
18            ProfileImpl p = new ProfileImpl("localhost",1099,"JADE");
19
20            //pour afficher l'interface graphique il faudra rajouter les param:
21            p.setParameter(Profile.LOCAL_HOST, "localhost");
22            p.setParameter(Profile.LOCAL_PORT, "1099");
23            p.setParameter(Profile.GUI, "true");
24
25            Object [] Arg1 = { "Pc", "500"};
26            Object [] Arg2 = {"Mouse", "45"};
27
28            ContainerController mc = rt.createMainContainer(p);
29
30            //AgentController ag1 = mc.createNewAgent("Buyer1","AgDisplay",null);
31            //AgentController ag2 = mc.createNewAgent("Buyer2","AgDisplay",null);
32
33            AgentController ag3 = mc.createNewAgent("Buyer3","AgDisplayA",Arg1);
34            AgentController ag4 = mc.createNewAgent("Buyer4","AgDisplayA",Arg2);
35
36            //ag1.start();
37            //ag2.start();
38            ag3.start();
39            ag4.start();
40        }
41        catch (Exception e)
42        {
43            System.out.println("Une erreur s'est produite : " + e.getMessage());
44            //e.printStackTrace();
45        }
46    }
47 }
```



Conclusion

Through these four exercises, we learned and manipulated some Jade operations including agent creation, argument passing to the agents and programmatically launching agents in order to gain familiarity.

Lab Report #03

Lab#03: JADE Communication

Introduction

On JADE communication lab#03, we explore the agent-to-agent communication through a series of exercises, we'll explore point-to-point messaging, message filtering, topic-based broadcasting, and serialization.

Exercise 1: (Point-to-point communication)

Let there be two agents (agent1, agent2). Give the code of each agent in JADE to carry out their following behaviour:

- Agent 2 sends a “Hello” message to Agent 1

- Agent 1 responds with a “Thank you” message

- Each agent displays the message received from the other agent.

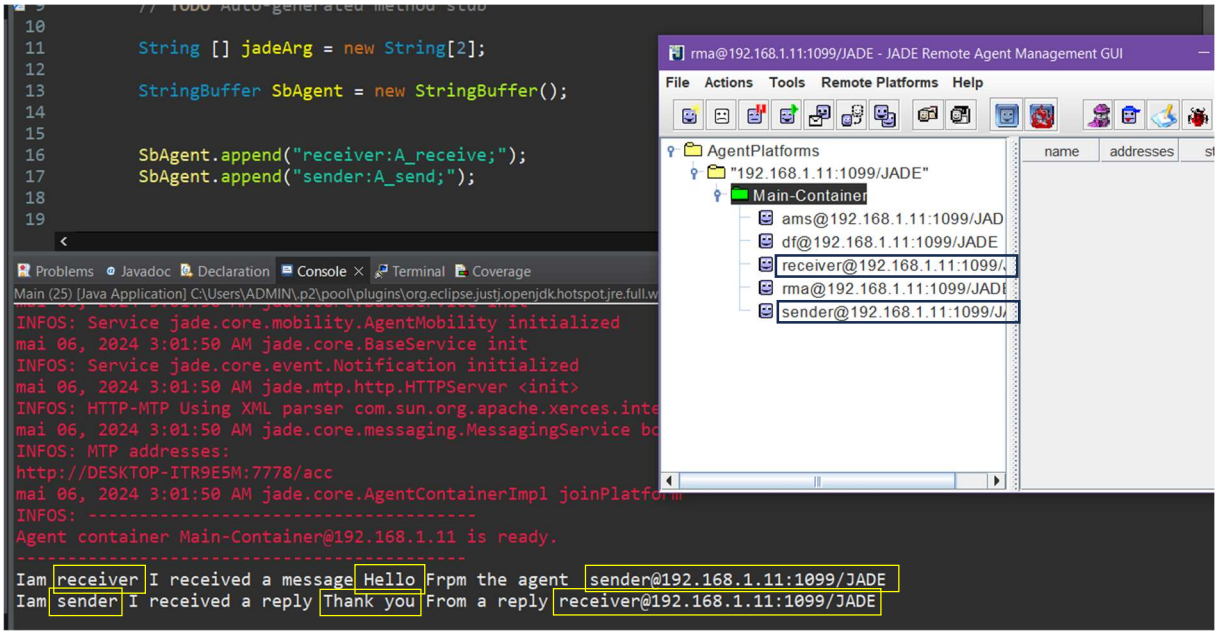
Solution

Please find the attached code in the **TechAgent_TP4_ACLMessages.zip** file.

```
public class A_receive extends Agent
{
    private static final long serialVersionUID = 1L;

    protected void setup()
    {

        addBehaviour(new CyclicBehaviour(this)
        {

            private static final long serialVersionUID = 1L;

            @Override
            public void action()
            {

                // TODO Auto-generated method stub
                MessageTemplate mt= MessageTemplate.MatchPerformative(ACLMessage.INFORM);

                ACLMessage msg= receive(mt);

                if(msg!=null)
                {
                    System.out.println("Iam "+ getLocalName() +" I received a message " + msg.getContent() +" Frpm the agent  "+ msg.getSender().getName());
                    ACLMessage reply=msg.createReply();

                    reply.setPerformative(ACLMessage.INFORM);
                    reply.setContent("Thank you");

                    send(reply);
                }
                block();
            }
        });
    }
}

```

```java
public class A_send extends Agent
{

    private static final long serialVersionUID = 1L;

    protected void setup()
    {
        ACLMessage msg=new ACLMessage(ACLMessage.INFORM);

        msg.setContent("Hello");
        msg.addReceiver(new AID("receiver",AID.ISLOCALNAME));

        send(msg);

        addBehaviour(new CyclicBehaviour(this)
        {

            private static final long serialVersionUID = 1L;

            @Override
            public void action()
            {
                // TODO Auto-generated method stub

                ACLMessage msg=new ACLMessage(ACLMessage.INFORM);

                msg=receive();

                if(msg!=null)
                {
                    System.out.println("Iam "+ getLocalName() +" I received a reply " + msg.getContent() +" From a reply "+ msg.getSender().getName());
                }
                block();
            }
        });
    }
}
```

As a result:



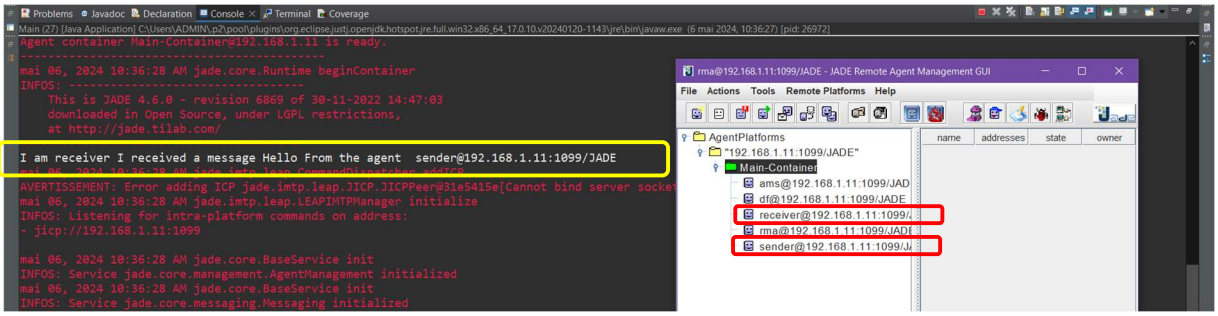
Exercise 2

a) Create the same receiver agent in main container.

Please find the attached code in the **TechAgent_TP3_ContainerA.zip** file.

```java
public class Main
{

    public static void main(String[] args)
    {
        String[] jadeArg = new String[2];
        StringBuffer SbAgent = new StringBuffer();

        SbAgent.append("receiver:a_receive;");
        SbAgent.append("sender:a_sender;");

        jadeArg[0] = "-gui";
        jadeArg[1] = SbAgent.toString();

        jade.Boot.main(jadeArg);

        // Creation of the receiver agent in the main container
        jade.core.Runtime rt = jade.core.Runtime.instance();
        jade.core.ProfileImpl p = new jade.core.ProfileImpl(false);

        p.setParameter(jade.core.Profile.MAIN_HOST, "localhost");

        jade.wrapper.AgentContainer mainContainer = rt.createMainContainer(p);
        try
        {
            jade.wrapper.AgentController receiver = mainContainer.createNewAgent("receiver", "a_receive", null);

            receiver.start();
        }
        catch (jade.wrapper.ControllerException e)
        {
            e.printStackTrace();
        }
    }
}
```

After starting the agents, it creates a new instance of a_receive agent and starts it in the main container. Like this, both sender and receiver agents are created and run in the main container.



b) Create a simple container and the same sender agent in it (by program).

Please find the attached code in the **TechAgent_TP3_ContainerB.zip** file.

We add a new class called simplecontainer in which we define the necessary steps for creating a simple container and the same sender agent in it.

```java
public class simplecontainer
{
    public static void main(String[] args)
    {
        try
        {
            Runtime rt=Runtime.instance();

            ProfileImpl p= new ProfileImpl(false);

            p.setParameter(Profile.MAIN_HOST, "localhost");

            AgentContainer container=rt.createAgentContainer(p);

            //container.start();

            AgentController ag2=container.createNewAgent("agent2","a_sender", null);

            ag2.start();
        }
        catch (ControllerException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

It creates a simple container, and within it, it instantiates and runs an agent of the class a_sender.

Exercise 4+5: (Broadcast Communication & Serialization)

Realise Topic-based communication between a sender agent that creates a topic « JADE » and a set of receiver agents that register to that topic.

Consider a product defined by a name and a price. Write the code of two agents (Seller and Buyer) who communicate as follows: the Seller agent creates a new product and sends it to the Buyer agent.

The Buyer agent displays the product received.

(Buyer and seller instead of sender and receiver)

Solution

We implemented the following java classes and run them as run configurations with the command:

-gui -services jade.core.messaging.TopicManagementService A0:A2Topic;A1:A1Topic;A2:A1Topic;A3:A1Topic

```
public class A1Topic extends Agent
{

    private static final long serialVersionUID = 1L;

    @Override
    protected void setup()
    {
        try
        {
            TopicManagementHelper TopicHelper = (TopicManagementHelper) getHelper(TopicManagementHelper.SERVICE_NAME);

            final AID topic = TopicHelper.createTopic("JADE");

            TopicHelper.register(topic);

            addBehaviour(new CyclicBehaviour(this)
            {

                private static final long serialVersionUID = 1L;

                public void action ()
                {
                    ACLMessage msg = receive(MessageTemplate.MatchTopic(topic));

                    if (msg!=null)
                    {
                        System.out.println("\nAgent "+getLocalName()+" : Message about topic "+topic.getLocalName()+" received contentent is : "+msg.getContent());
                    }
                    else
                    {
                        block();
                    }
                }
            });
        }
        catch (ServiceException e)
```

A1Topic.java | A2Topic.java | Buyer.java | Main.java | Product.java | Seller.java

```java
public class A2Topic extends Agent
{

    private static final long serialVersionUID = 1L;

    @Override
    protected void setup()
    {
        try
        {
            TopicManagementHelper TopicHelper = (TopicManagementHelper) getHelper(TopicManagementHelper.SERVICE_NAME);

            final AID topic = TopicHelper.createTopic("JADE");

            //TopicHelper.register(topic);

            addBehaviour(new TickerBehaviour(this, 5000)
            {

                private static final long serialVersionUID = 1L;

                public void onTick ()
                {
                    System.out.println("\nAgent "+getLocalName()+" : Sending message about topic "+topic.getLocalName());

                    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);

                    msg.addReceiver(topic);

                    msg.setContent(String.valueOf(getTickCount()));

                    send(msg);
                }
            });
        }
        catch (Exception e)
        {
```

```java
public class Buyer extends Agent
{

    private static final long serialVersionUID = 1L ;

    @Override
    protected void setup()
    {
        ACLMessage m;

        Product P;

        m = receive();

        if(m!=null)
        {
            try
            {
                P= (Product)m.getContentObject();

                System.out.println("I am "+getLocalName()+" I received the product "+P.name+" with the price : "+P.price);
            }
            catch (Exception e)
            {

            }
        }
    }
```

```java
public class Seller extends Agent
{
    private static final long serialVersionUID = 1L ;


    Product P = new Product() ;

    ACLMessage m;

    @Override
    protected void setup()
    {
        P.price = 2500 ;
        P.name = "Keyboard" ;

        m = new ACLMessage(ACLMessage.INFORM);

        m.addReceiver ( new AID("Buyer",AID.ISLOCALNAME));



        try
        {
            m.setContentObject(P);

            m.setLanguage("JavaSerialization");
            //le msg s'envoie comme une chaine de caractere puis il se reconstitue lors de son arrivé en forme d'objet Produit !

            send(m);
        }
        catch (Exception e)
        {
```

Writable | Smart Insert | 43 : 9 : 723

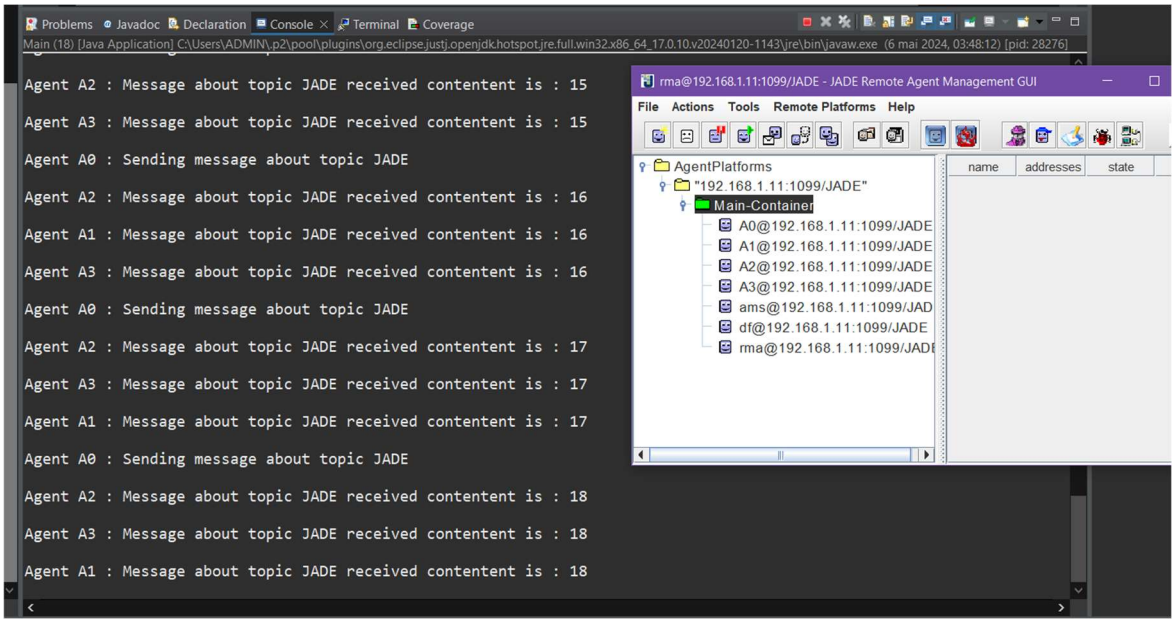
A1Topic.java | A2Topic.java | Buyer.java | Main.java | Product.java | Seller.java

```
1  import java.io.*;
2
3
4  public class Product implements Serializable
5  {
6
7      //private static final long serialVersionUID = 1L;
8
9      double price;
10     String name;
11 }
12
13
```

Problems | Javadoc | Declaration | Console | Terminal | Coverage

Main (18) [Java Application] C:\Users\ADMIN\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\

```
mai 06, 2024 3:48:13 AM jade.tools.ToolAgent init
AVERTISSEMENT: NotificationService not installed. Some tool may not work properly.

Agent A0 : Sending message about topic JADE

Agent A2 : Message about topic JADE received contentent is : 1

Agent A3 : Message about topic JADE received contentent is : 1

Agent A1 : Message about topic JADE received contentent is : 1

Agent A0 : Sending message about topic JADE

Agent A2 : Message about topic JADE received contentent is : 2

Agent A3 : Message about topic JADE received contentent is : 2

Agent A1 : Message about topic JADE received contentent is : 2
```



Please find the attached code in the **TechAgent_TP5_ACLMessages.zip** file.

We combined in this code both broadcast communication and serialization.

As we can notice, in one hand, the agents A1Topic and A2Topic are participating in topic-based communication: A1Topic receives messages published to that topic by other agents while A2Topic publishes messages about the topic using a TickerBehaviour which allows the broadcasting.

In the other hand, the Product class indicates that elements of this class can be serialized, for example: the Seller agent which is an instance of the Product class (P) is serialized and sent as the content of an ACLMessage to the Buyer agent. Upon receiving the message, the Buyer agent deserializes the content back into a Product object and processes it.

Conclusion

In this lab#03 work, we learned the ACLMessage mechanism and some functions related to it like: Point-to-point communication and Broadcast communication through topic-based messaging, we explored also the Container concept and Serialization for transmitting complex data objects between agents.

Lab Report #04

Lab#04: Behaviours

Introduction

In this lab we focused on understanding and implementing various types of behaviors in JADE agents. we learned about agent migration, primitive behaviors, and composite behaviors to understand how agents can perform actions autonomously and interact with their environment.

Exercise 1: (Agent Migration: Mobile Agent)

Create an agent under a container (container_2). Migrate it through JADE GUI (Interface) to MainContainer by displaying a message before, during and after moving.

Solution

```java
public class simplecontainer
{

    public static void main(String[] args)
    {
        // Get a hold of the JADE runtime
        Runtime rt = Runtime.instance();

        // Create a default profile for the main container
        Profile pMain = new ProfileImpl(true); // true means it's the main container
        pMain.setParameter(Profile.MAIN_HOST, "localhost");

        // Create and start the main container
        AgentContainer mainContainer = rt.createMainContainer(pMain);

        try
        {
            // Start the main container
            mainContainer.start();

            // Create a profile for the additional container
            ProfileImpl pContainer = new ProfileImpl(false); // false means it's not the main container
            pContainer.setParameter(Profile.MAIN_HOST, "localhost");

            // Create the additional container
            AgentContainer container = rt.createAgentContainer(pContainer);

            // Create and start agents in the additional container
            AgentController agent1 = container.createNewAgent("agent1", "BehaviourComplex", null);
            AgentController agent2 = container.createNewAgent("agent2", "BehaviourComplex", null);

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

```java
protected void takeDown()
{
    System.out.println("agent terminated");
}
public void DoMove(Location L) {
    System.out.println("Agent migrration"+L.getName());
}
protected void beforeMove() {
    System.out.println("before miggration"+ this.getAID().getName());

    try
    {
        System.out.println("from container"+this.getContainerController().getContainerName());
    }
    catch (ControllerException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
```

```java
protected void afterMove()
{
    System.out.println("after miggration"+ this.getAID().getName());
    try
    {
        System.out.println("to the container "+ this.getContainerController().getContainerName());
    }
    catch (ControllerException e)
    {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Exercise 2: (Primitive Behaviour)

Create an agent running several simple behaviours. Test these behaviours each alone: Behavior, OneshotBehaviour, CyclicBehaviour, WakerBehaviour and TickerBehaviour. Each behaviour displays a simple message.

Solution

1. OneshotBehaviour

```java
//————————————————————OneShotBehaviour————————————————————//

addBehaviour(new OneShotBehaviour()
{


    private static final long serialVersionUID = 1L;

    @Override
    public void action()
    {
        // TODO Auto-generated method stub
        System.out.println("this action runs 1 time");
    }
});
```

This behavior runs its action method only once. In this case, it prints a message.

2. CyclicBehaviour

```
//————————————————cyclicBehaviour————————————————————————
addBehaviour(new CyclicBehaviour()
{

    private static final long serialVersionUID = 1L;

    @Override
    public void action()
    {
        // TODO Auto-generated method stub
        System.out.println("this action runs an infinity times" );
    }
});
```

This behavior runs indefinitely, continuously executing its action method and printing a message.

3. WakerBehaviour

```
//————————————————WakerBehaviour————————————————————————//
//ce reveille apres certain temps et fait une action
addBehaviour(new WakerBehaviour(this,6000)
{


    private static final long serialVersionUID = 1L;

    public void onWake() {
        System.out.println(getLocalName()+"i'm daying");
        doDelete();
    }
});
```

This behavior wakes up after a specified delay (6000 milliseconds) and performs an action. Here, it prints a message and then deletes the agent using doDelete().

4. TickerBehaviour

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


    private static final long serialVersionUID = 1L;

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

    }
});
```

This behavior executes its action method at regular intervals (every 30 seconds in this case), printing a message each time.

Exercise 3: (Composite behaviour)

Create and launch 2 agents that owns behaviors executing in parallel. These behaviors can be activated 3 times and display greetings (Hello) in several languages.

Example of requested result :

I, Agent a1, my address is (agent-identifier :name a1@192.168.1.4:1099/JADE :addresses

(sequence http://Admin:7778/acc))

I, Agent a2, my address is (agent-identifier :name a2@192.168.1.4:1099/JADE :addresses

(sequence http://Admin:7778/acc))

I execute several behaviors in parallel

I execute several behaviors in parallel

(2 -> 1/3) عليكم السالم a

(1 -> 1/3) عليكم السالم a

a1 -> bonjour (1/3)

a1 -> hallo (1/3)

a1 -> buongiorno (1/3)

a1 -> buenos dias (1/3)

a1 -> Olá (1/3)

a2 -> bonjour (1/3)

a2 -> hallo (1/3)

a1 -> saluton (1/3)

(1 -> 2/3) عليكم السالم a

a1 -> bonjour (2/3)

a1 -> hallo (2/3)

a2 -> buongiorno (1/3)

a2 -> buenos dias (1/3)

a1 -> buongiorno (2/3)

a1 -> buenos dias (2/3)

a2 -> Olá (1/3)

a1 -> Olá (2/3)

a2 -> saluton (1/3)

(2 -> 2/3) عليكم السالم a

a1 -> saluton (2/3)

Solution

In this Java program we create two agents (agent1 and agent2). Each agent executes a set of behaviors at the same time, and each behavior is responsible for printing a greeting message in different languages. These behaviors are activated three times before they terminate.

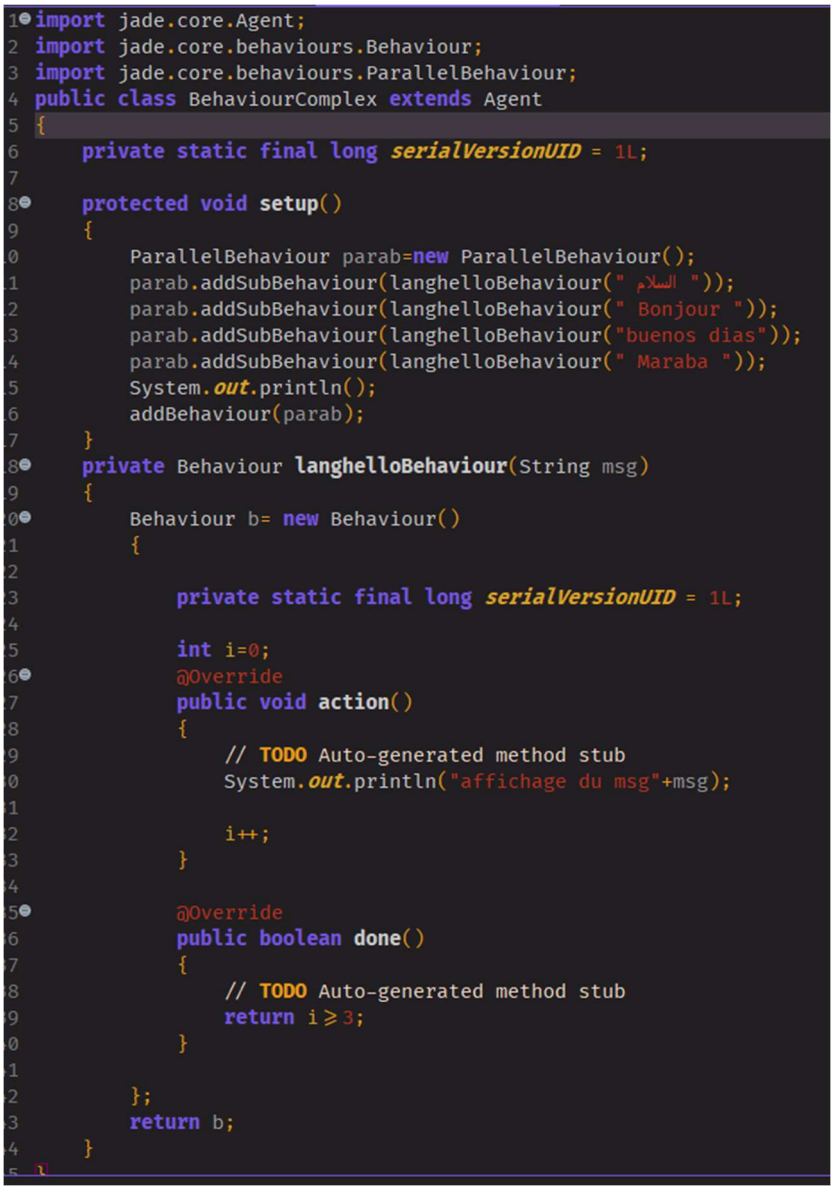
```
public class Main
{

    public static void main(String[] args)
    {
        // TODO Auto-generated method stub



        String [] jadeArg = new String[2];
        StringBuffer SbAgent = new StringBuffer();


        SbAgent.append("agent:BehaviourComplex;");
        SbAgent.append("agent1:BehaviourComplex;");

        jadeArg[0]="-gui";

        jadeArg[1]=SbAgent.toString();

        jade.Boot.main(jadeArg);


    }

}
```

Initialization: The main method initializes an array jadeArgs to hold JADE command-line arguments.

Agents Definition: A StringBuffer SbAgent is used to define two agents (agent and agent1), both of type BehaviourComplex.

Arguments Assignment: The first element of jadeArgs is set to -gui, which instructs JADE to start the GUI. The second element is the agent definition string.

JADE Boot: jade.Boot.main(jadeArgs) launches the JADE runtime with the specified arguments, starting the agents.



```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.ParallelBehaviour;
public class BehaviourComplex extends Agent
{
    private static final long serialVersionUID = 1L;

    protected void setup()
    {
        ParallelBehaviour parab=new ParallelBehaviour();
        parab.addSubBehaviour(langhelloBehaviour(" السلام "));
        parab.addSubBehaviour(langhelloBehaviour(" Bonjour "));
        parab.addSubBehaviour(langhelloBehaviour("buenos dias"));
        parab.addSubBehaviour(langhelloBehaviour(" Maraba "));
        System.out.println();
        addBehaviour(parab);
    }
    private Behaviour langhelloBehaviour(String msg)
    {
        Behaviour b= new Behaviour()
        {

            private static final long serialVersionUID = 1L;

            int i=0;
            @Override
            public void action()
            {
                // TODO Auto-generated method stub
                System.out.println("affichage du msg"+msg);

                i++;
            }

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

        };
        return b;
    }
}
```

Agent Setup: The setup method is called when the agent is initialized.

ParallelBehaviour: A ParallelBehaviour parab is created to allow multiple behaviors to run concurrently.

Adding Sub-Behaviours: The parab object has several sub-behaviours added to it, each created by the langhelloBehaviour method, which takes a greeting message as a parameter.

Print Agent Information: The agent prints its name and address, indicating it is executing multiple behaviors in parallel.

Adding Behaviour: The parab (which includes all the sub-behaviours) is added to the agent.

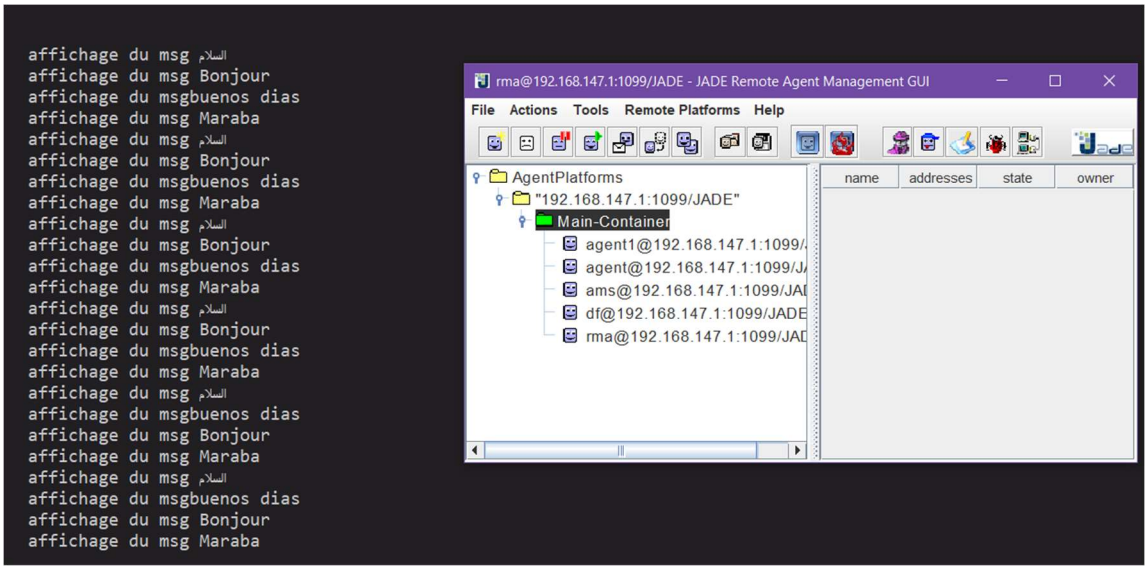
langhelloBehaviour Method

Behaviour Definition: The method returns a new Behaviour object.

Action Method: The action method prints the agent's name, the message, and the current iteration number (formatted as (i+1) to start from 1).

Done Method: The done method returns true when the message has been printed three times, stopping the behavior.

Output:



In this code we created and run multiple agents in JADE, each with multiple concurrent behaviors. Each behavior repeatedly prints a hello message three times.

Conclusion

Through these exercises, we have explored different aspects of agent behaviors in JADE. We learned how to migrate agents between containers, implemented and tested various primitive behaviors, and created agents with composite behaviors that run concurrently.