

COS30018 – Intelligent Systems

Week 2 tutorial

This week we will cover the following items:

- The basics of Choco
- JADE: Agent communications
- JADE: Creating a distributed platform & running multiple JADE platforms
- JADE: Create your custom agent

1. The basics of Choco

As discussed during the lecture, Constraint Satisfaction Problems (CSP) define a set of states as assignments of values to variables from a domain. Constraints then encode the allowable assignments of values to these variables: thus, constraints restrict the set of all states to a set of possible solutions. A solver is then used to conduct a sophisticated search of the problem space and find a solution. A Constraint Optimization Problem (COP) is a problem where each potential solution has a weight/cost/value associated with it and the solver seeks to minimize or maximize this value. This week we will talk about one CSP/COP solving library called Choco.

In the tutorial this week, we try to solve the n-queens problem using Choco, you can find the detailed instructions in the file “The basics of Choco” on Canvas. If you are interested in learning more about Choco you should start with this: <https://choco-solver.org/docs/getting-started/>

After reading through the instruction material, and running the code you should try working through the Traveling Salesman Problem tutorial here: <https://choco-solver.org/tutos/traveling-salesman-problem/>; this will be very useful for those of you doing the VRP assignment.

Some short notes to get started:

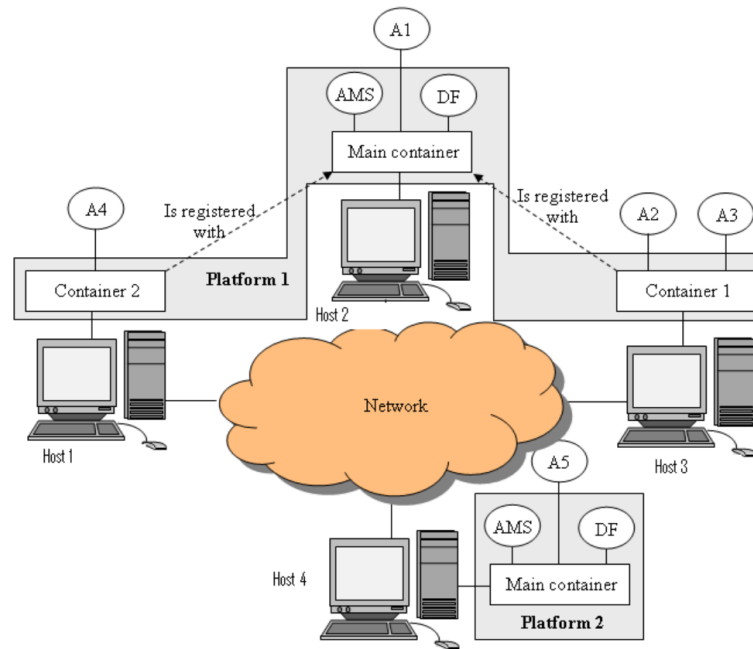
Choco Model — The main object you will use when defining a Choco Constraint Problem is the Model object. You define your problem by creating one of these objects and then using it to create your problem variables and constraints.

Variables — Choco has several variable types (see <https://javadoc.io/static/org.choco-solver/choco-solver/4.10.3/org.chocosolver.solver/org/chocosolver/solver/variables/package-summary.html>) but you will probably just need to use either IntVar or BoolVar. *You should always use your model object to create your variables so that the model has a reference to the variable.*

Constraints — Choco has several types of constraints (see <https://javadoc.io/static/org.choco-solver/choco-solver/4.10.3/org.chocosolver.solver/org/chocosolver/solver/constraints/IConstraintFactory.html>) but you will probably find the arithm and allDif constraints used in the examples are enough for most problems. *You should always use your model object to create your constraints so that the model has a reference to the constraints. Also, REMEMBER to use POST(): if you don't call post() on a constraint it will not get enforced during the solving process.*

2. Agent communication in JADE

Agents can communicate regardless of whether they live in the same container or in different containers, or even in different platforms.



They communicate by sending & receiving messages. The messages sent and received by agents contain these fields:

- The sender: from who?
- The receiver: to who?
- The communicative act: type of the message (e.g., INFORM, REQUEST, ...)
- The content: the information conveyed by the message

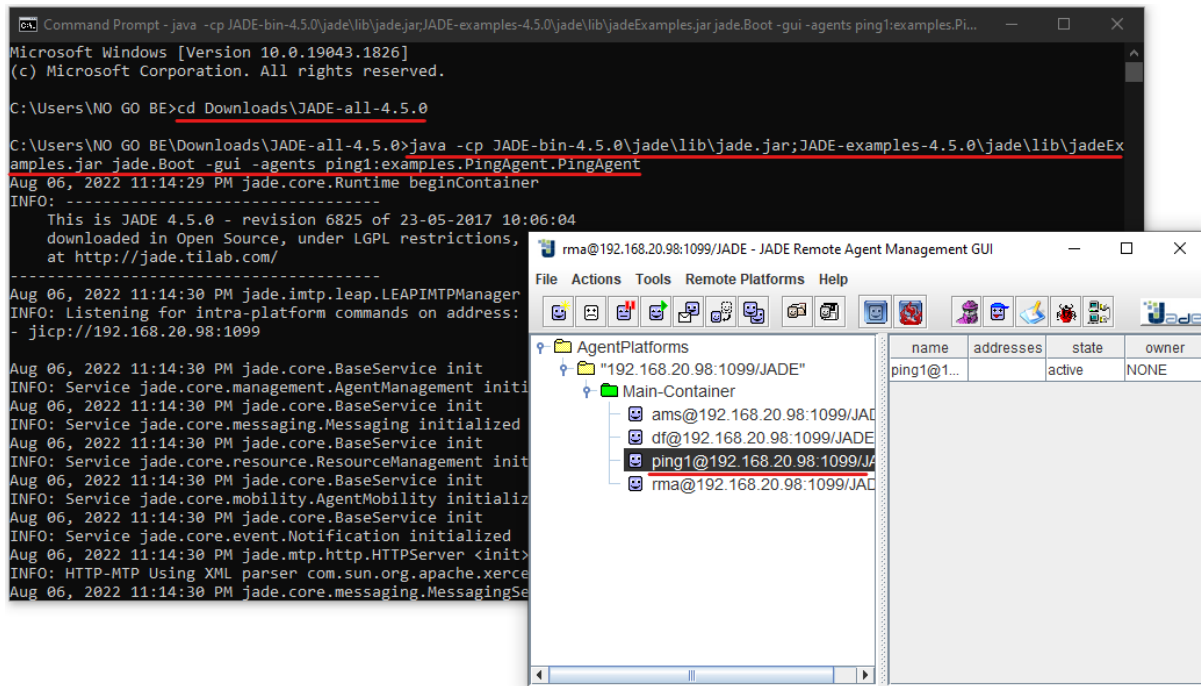
In this section we will start some agents and we will make them communicate. We will use the `DummyAgent`, a ready-made agent (included among the JADE tools) that can be used to send custom messages (it can also receive messages), and the `PingAgent`, a very simple agent waiting for messages. If a `REQUEST` message with content “ping” is received, an `INFORM` message with content “pong” is sent back. If any other message is received, a `NOT_UNDERSTOOD` message is sent back. The `PingAgent` must be compiled first.

a. Compiling the PingAgent

`PingAgent` is pre-written by JADE and can be found in the `JADE-examples-4.5.0\jade src\examples\PingAgent`.

To compile it, you just need to type the following command in your command line (similar to how to start a “HelloWorld” agent demonstrated in week 1) and you will have a `PingAgent` shown as below:

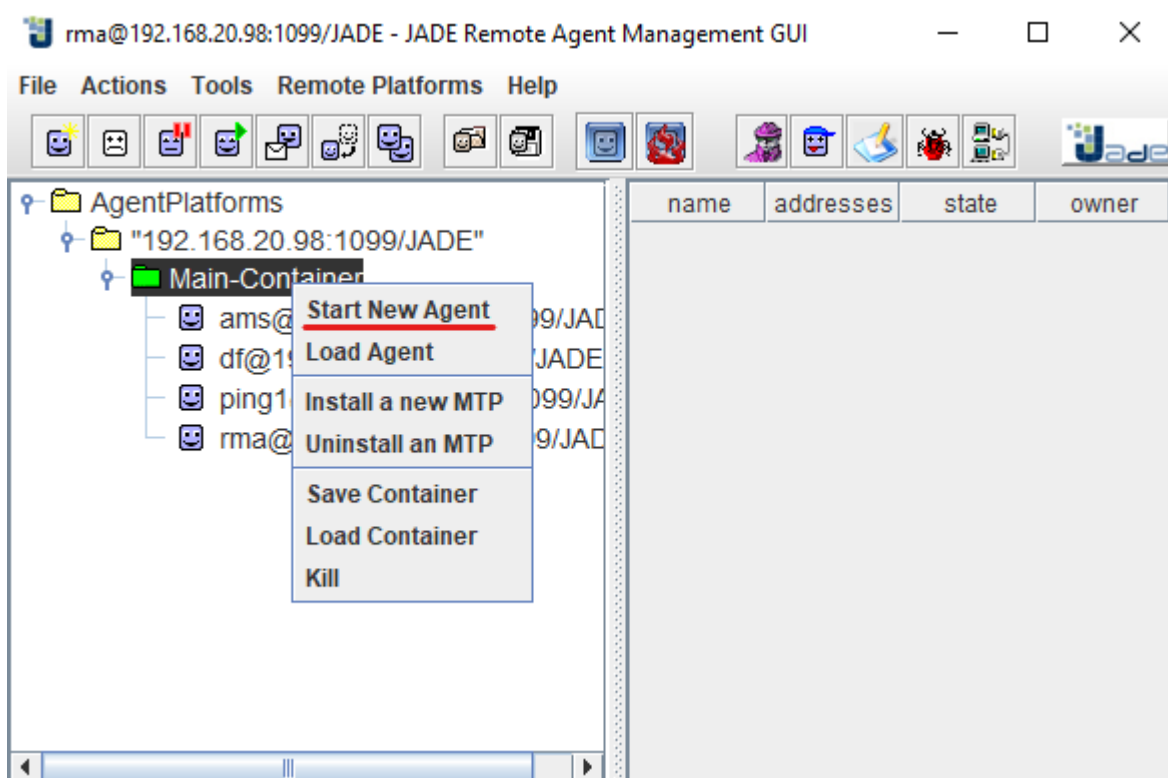
```
java -cp JADE-bin-4.5.0\jade\lib\jade.jar;JADE-examples-4.5.0\jade\lib\jadeExamples.jar jade.Boot -gui -agents ping1:examples.PingAgent.PingAgent
```



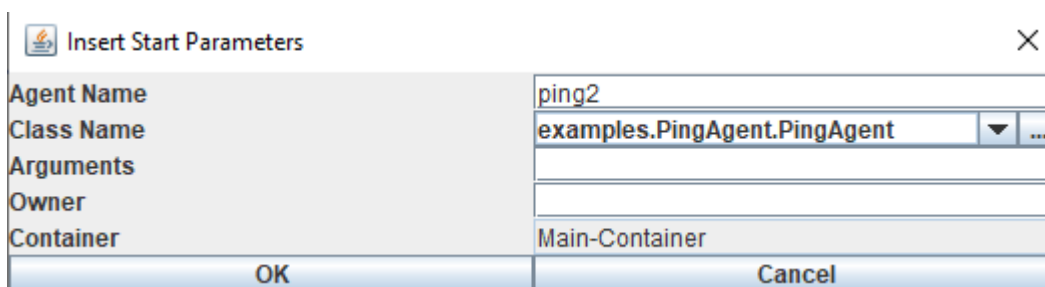
Note:

- “cp” is short for CLASSPATH, and “-cp” allows us to access to a specific function in a given class.
- We use “jade.Boot -gui” from “jade.jar” class to start the JADE environment and the GUI.
- We use “-agents” to create an agent.
- “ping1” is the name of the agent, we can name it anything.
- “examples.PingAgent.PingAgent” is the example PingAgent that is included in the JADE installation files.

Let’s create our second PingAgent using the GUI. Right-click on the Main Container and choose “Start New Agent”:



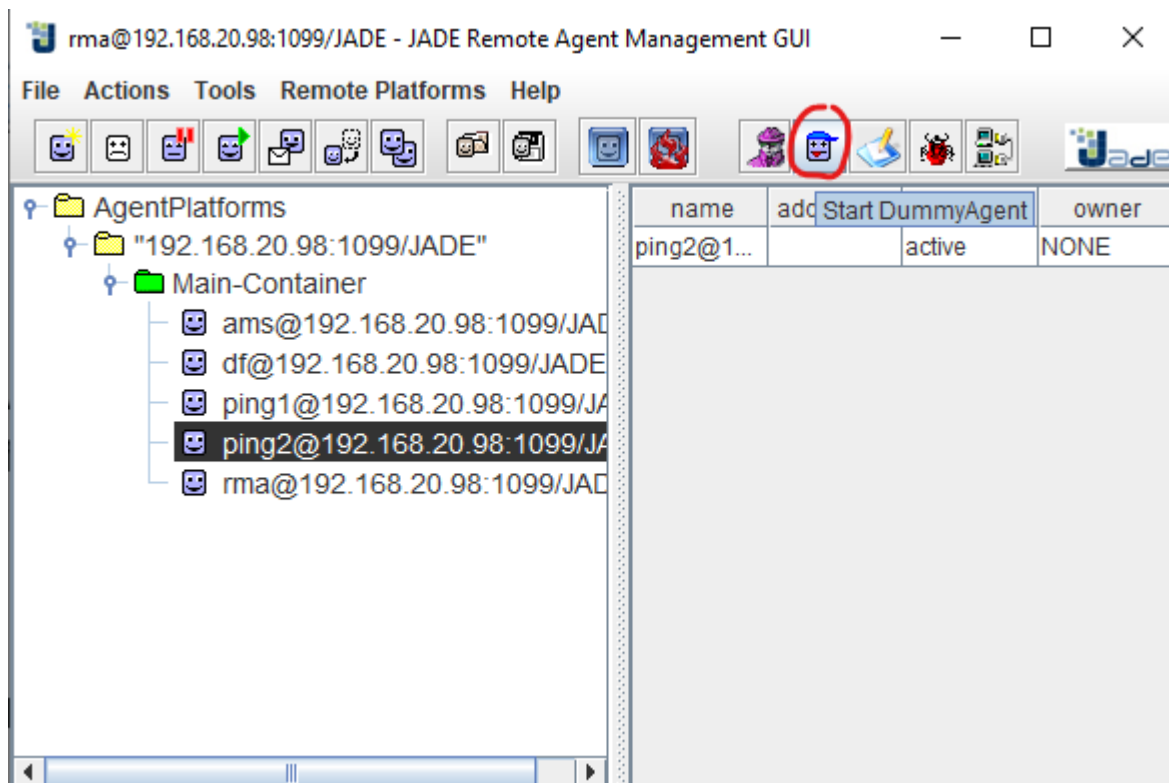
In the popup menu, type “ping2” for the Agent Name and “examples.PingAgent.PingAgent” for the Class Name. Then click OK:



The “ping2” agent should appear in the GUI under the Main Container.

b. Compile the DummyAgent

At this point, we have two PingAgents waiting for messages. Now we will create a DummyAgent to send messages to them. We can create a DummyAgent directly from the GUI by clicking “start DummyAgent” button:



Then select the REQUEST communicative act, type "ping" in the Content:

da0@192.168.20.98:1099/JADE - DummyAgent

General Current message Queued message

ACLMessage Envelope

Sender: Set da0@192.168.20.98:1099/JAC

Receivers:

Reply-to:

Communicative act: request

Content:
ping

Language:

Encoding:

Ontology:

Protocol: Null

Conversation-id:

In-reply-to:

Reply-with:

Reply-by: Set

User Properties:

And select a receiver for the message by right-clicking in the Receivers text area and select the Add menu item, type "ping1" in the NAME:

AID ×

NAME ☒ ping1

Addresses

Resolvers










Properties

OK Cancel

Finally click the button with the message icon in the DummyAgent GUI:

da0@192.168.20.98:1099/JADE - DummyAgent — □ ×

General Current message Queued message

ACLMessage Envelope

Sender: da0@192.168.20.98:1099/JAC

Receivers: ping1@192.168.20.98:1099/JADE

Reply-to:

Communicative act: request

Content: ping

Language:

Encoding:

Ontology:

Protocol: Null

Conversation-id:

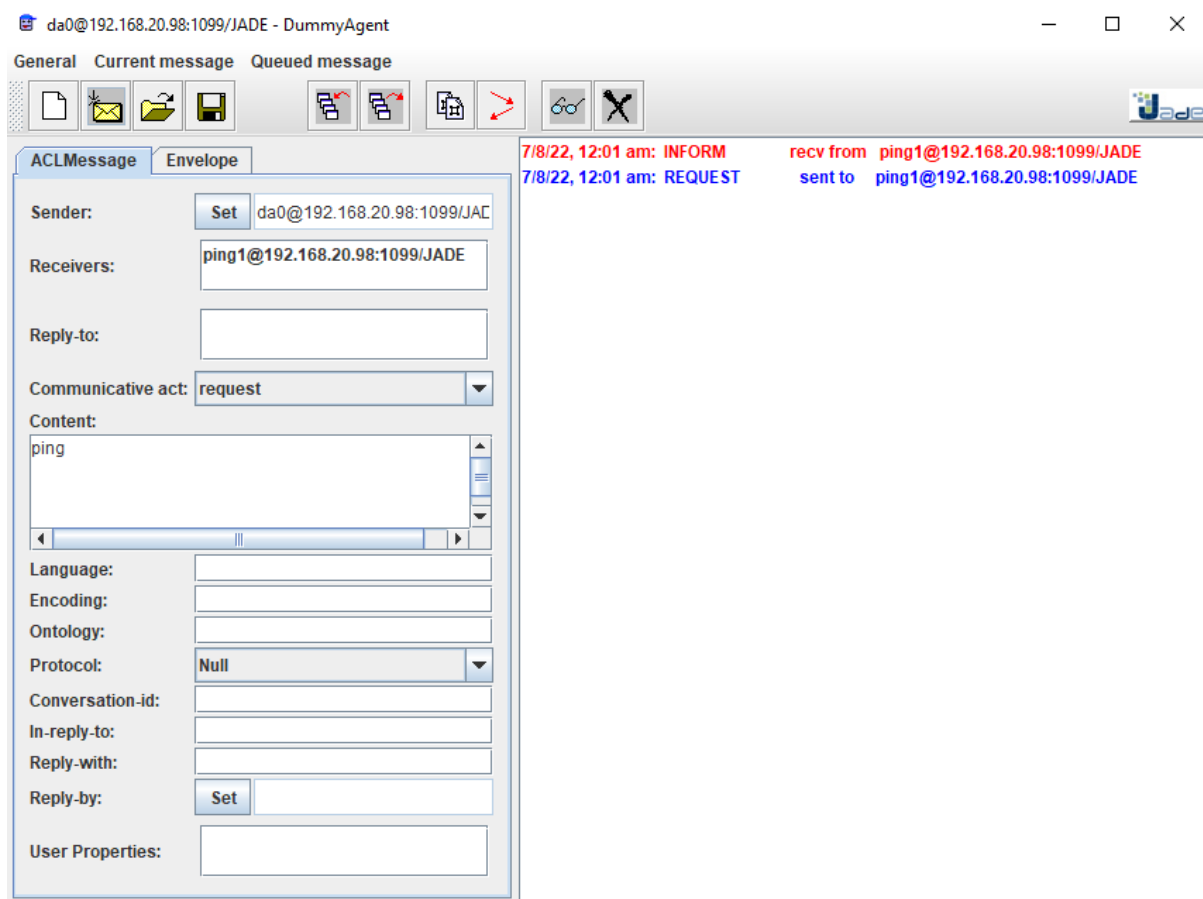
In-reply-to:

Reply-with:

Reply-by:

User Properties:

In the right pane of the DummyAgent GUI two lines appear, one red, the other blue. The most recent is the topmost. Blue refers to sent messages, red to received messages. You should see something similar to the image below:



Please keep your Main Container on to continue with part 3 below.

3. Create a JADE distributed platform & running multiple JADE platforms

a. Create JADE distributed platform:

In the last week, we saw how to start the JADE Main Container on your local machine. In this tutorial we will see how to start peripheral containers thus creating a distributed JADE platform.

Multiple containers on 1 computer:

Keep the Main Container from part 2 on, then open other shell, cd to the JADE-bin-4.5.0 file directory and type:

```
java -cp jade\lib\jade.jar jade.Boot -container
```

```

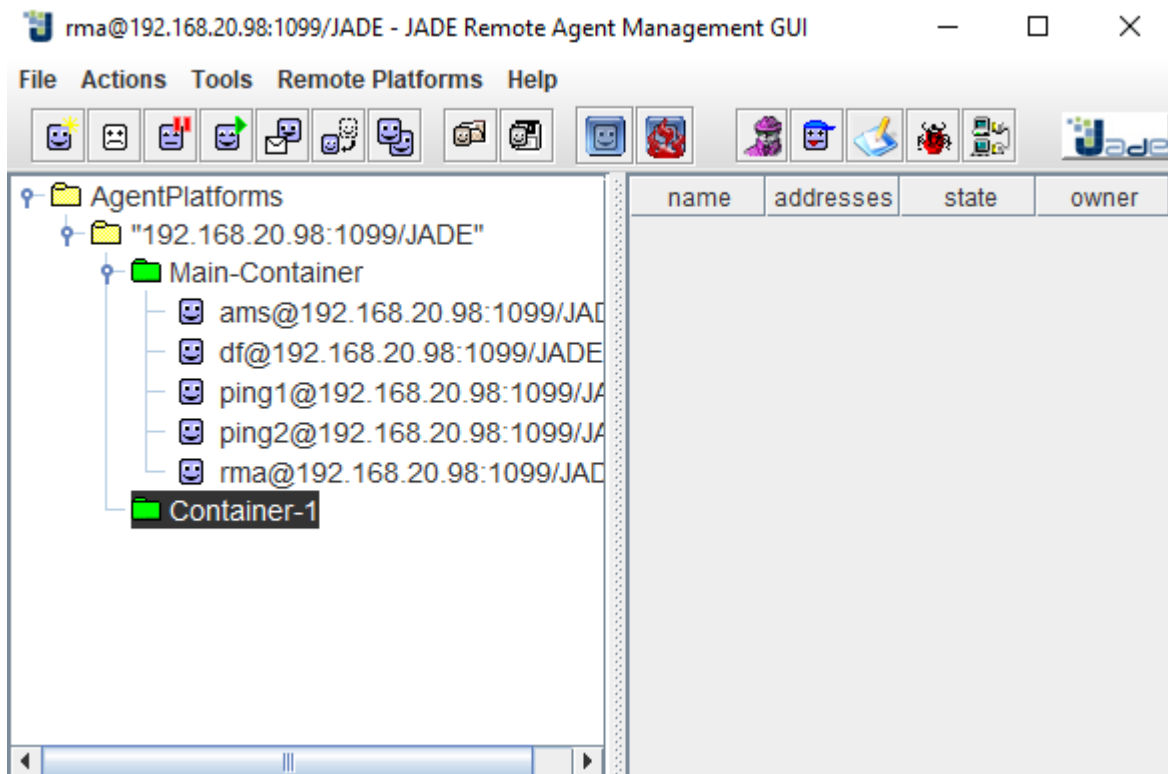
Command Prompt - java -cp jade\lib\jade.jar jade.Boot -container
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\NO GO BE>cd Downloads\JADE-all-4.5.0\JADE-bin-4.5.0

C:\Users\NO GO BE\Downloads\JADE-all-4.5.0\JADE-bin-4.5.0>java -cp jade\lib\jade.jar jade.Boot -container
Aug 07, 2022 12:29:59 AM jade.core.Runtime beginContainer
INFO: -----
This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----

```

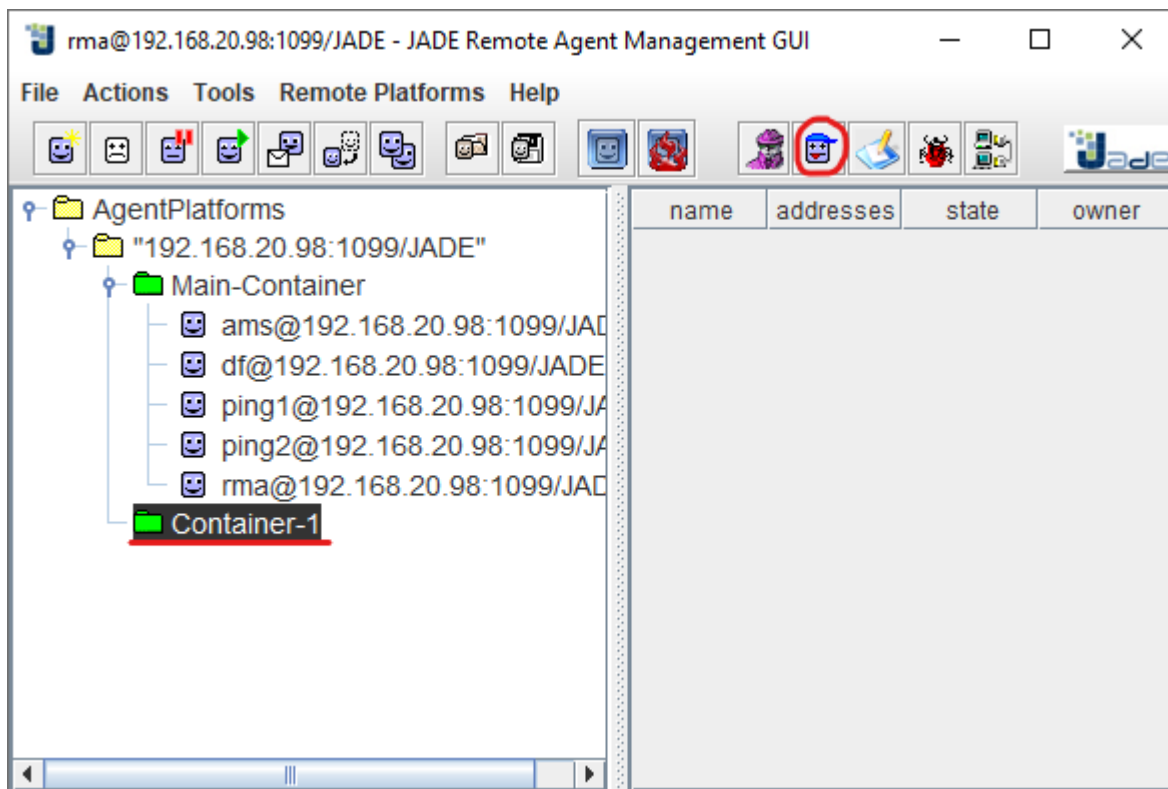
Now you will see a new container Container-1 is created beside the Main Container in your GUI, as depicted below:



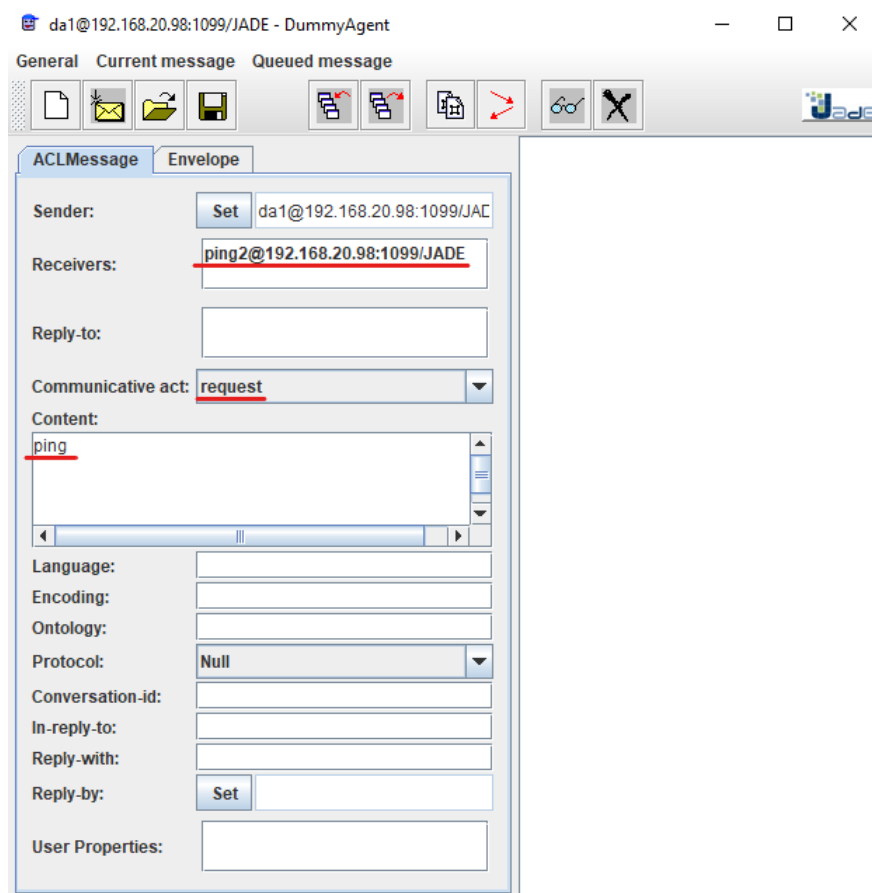
The `-container` option tells JADE to start a peripheral container instead of a Main Container. A peripheral container cannot live “alone” and must connect to a Main Container at startup. Therefore, typically it is necessary to tell it where to find the Main Container to connect to. This is done by using `-host <host-name>` and `-port <port-number>`. In our case it was *not* necessary to specify such options as the Main Container is running on the local host and on the JADE default port 1099. Therefore, the command line we typed is fully equivalent to:

```
java -cp jade\lib\jade.jar jade.Boot -container -host jade.tilab.com -port 1099
```

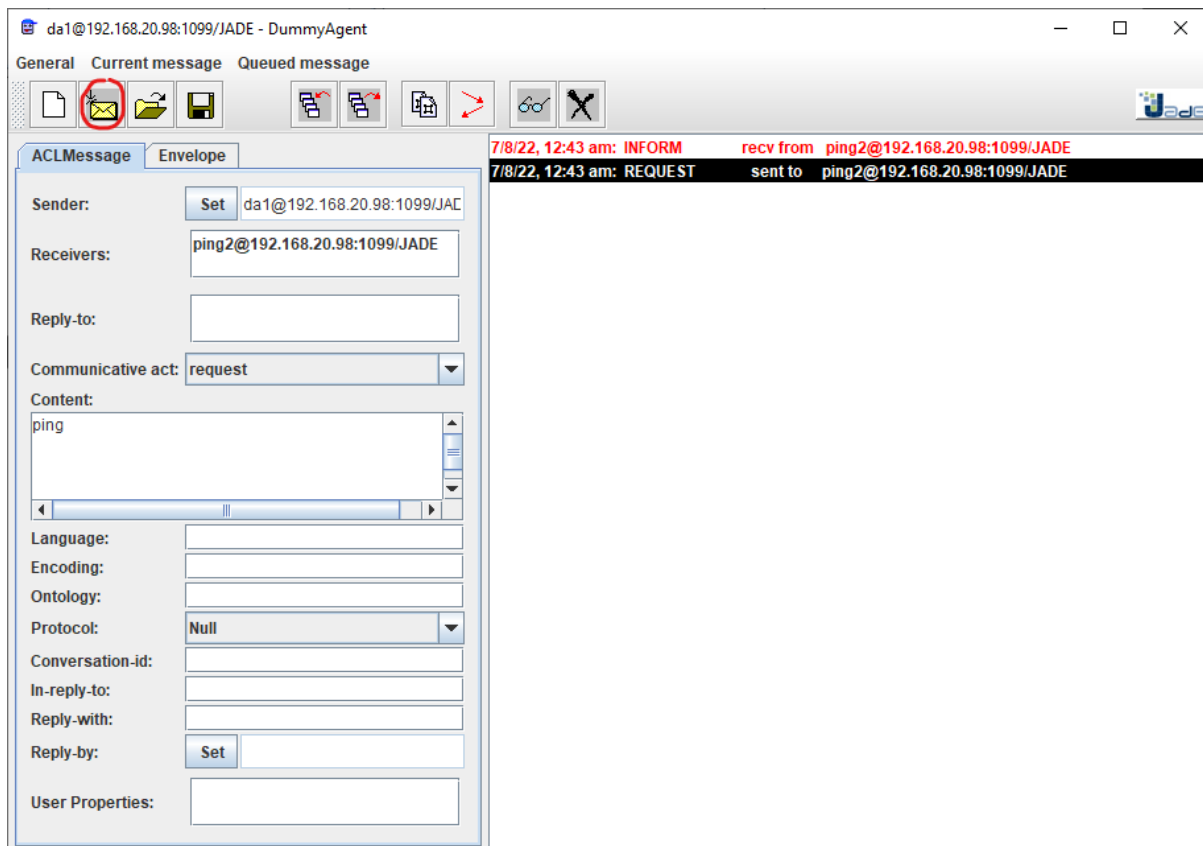
Now, let’s create a sample dummy agent in the newly created container to test if agents from different containers can communicate to each other. From the GUI, click on the “start DummyAgent” button (make sure you are selecting Container-1):



Now let's try to send a "ping" request from DummyAgent in Container-1 to ping2 PingAgent in Main Container by putting in these information:



Hit the send message button, you will see ping2 receives the REQUEST message and replies with an INFORM message on the right pane:



This demonstrates how agents from different containers can communicate to each other.

Remote containers:

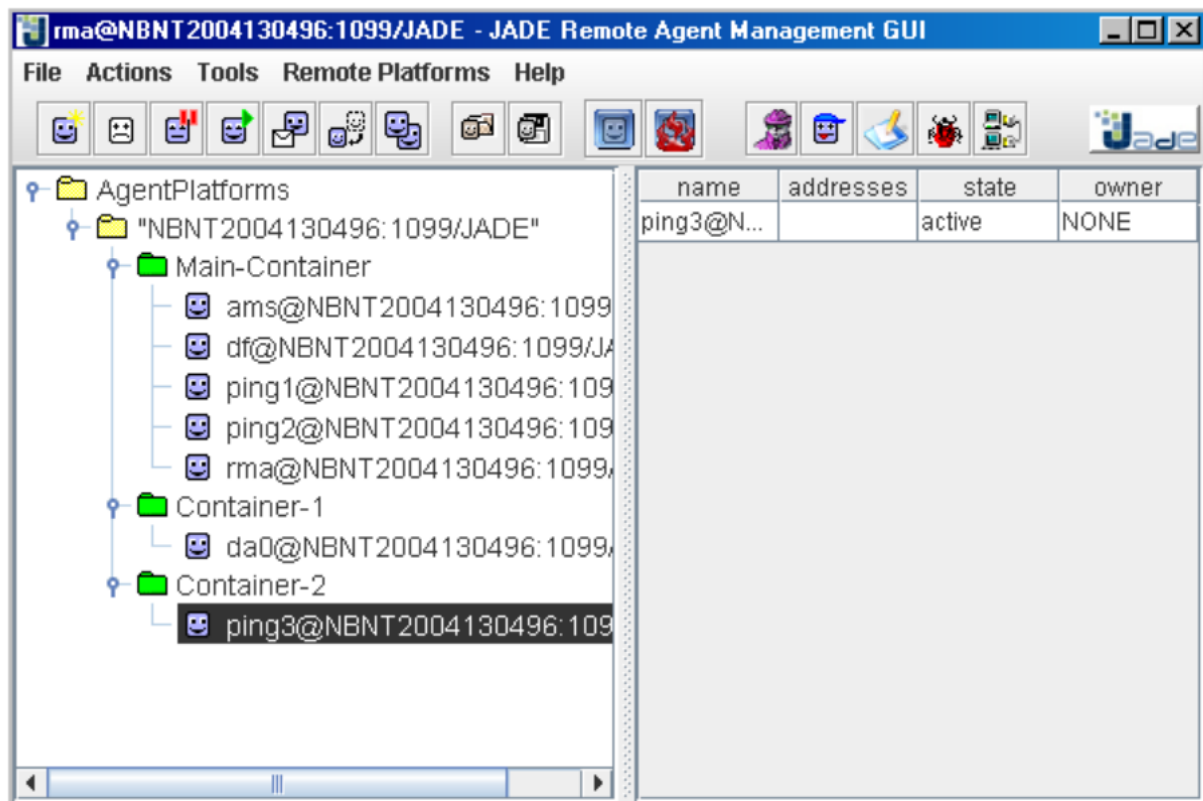
Up to now we created a JADE platform distributed over two processes running in the same host. Now we will create another container on a second computer thus obtaining a fully distributed platform.

For the sake of simplicity let's assume the host where we worked up to now (i.e. the one we started our Main Container and our peripheral container Container-1 on) is called **Computer1**. Let's also assume we have a second host called **Computer2** networked with the first one. Move to **Computer2** and unzip the JADE binary, source and examples distribution somewhere as you did in **Computer1**.

Then launch a JADE runtime on **Computer2**, and create a new peripheral container connecting to the Main Container in **Computer1** with a third PingAgent called ping3 on top by typing the command line below

```
java -cp jade\lib\jade.jar jade.Boot -container -host Computer1 -port 1099 -agents ping3:examples.PingAgent.PingAgent
```

The usual JADE runtime output saying that a container called Container-2 is up and running should appear. Looking at the GUI in **Computer1** you should see a situation similar to what depicted in image below:



You can now use the DummyAgent in **Computer1** send a message to the ping3 PingAgent in **Computer2** and check that ping3 replies as expected.

b. Running multiple JADE platforms

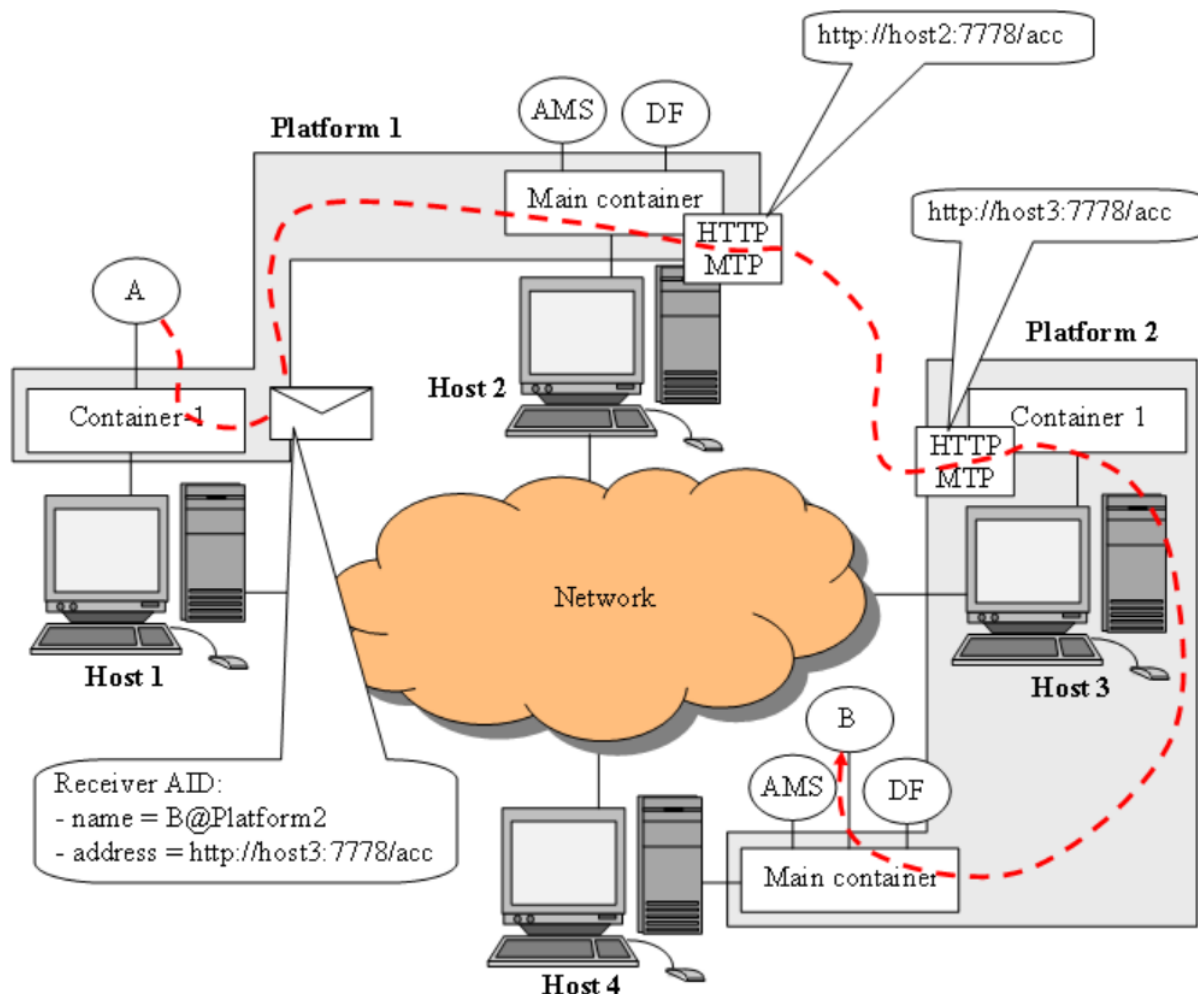
Until now, we saw a single JADE platform consisting of one Main Container and a number of peripheral containers running locally as well as on remote computers. We have also seen that agents in the same platform (no matter if they are in the same or in different containers) can communicate to each other. In this part, we will see how to start multiple JADE platforms and we will make agents belonging to different platforms communicate as well.

MTP:

Inter-platform communication, i.e. communication between agents living on different platforms, is based on modules called MTP (Message Transport Protocol).

MTP modules can be installed in whatever container and a container can have zero or more MTP modules installed in it. By default, JADE installs the HTTP MTP in a Main Container and no MTP on peripheral containers. However, you can change this default by manually configuring the HTTP MTP. Please refer to this link for more information: <https://jade.tilab.com/documentation/tutorials-guides/configuring-the-jade-http-mtp/>.

Agents running in a Container with no MTP are able to communicate with agents in remote platform anyway since JADE routes messages directed to foreign agents to the first container hosting a suitable MTP as depicted below:



Message from agent A/Container-1 (Platform 1) is first sent to the Main Container (Platform 1) because that is where we have the HTTP MTP is installed. Then the message is then sent to Container-1 (Platform 2) before reaching to the destination at agent B/Main Container (Platform 2).

Starting 2 platforms:

Let's assume we have the two network-connected hosts mentioned in part 3a (remote containers): **Computer1** and **Computer2** (of course you can do the same experiment on a single computer; in this case remember to specify different local ports for the Main Containers of the 2 platforms that is shown below).

If you are using only 1 single computer:

- Start the first platform `Platform1` by typing in the command line:
`java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform1`

```

Command Prompt - java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform1
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\N0 GO BE>cd Downloads\JADE-all-4.5.0\JADE-bin-4.5.0
C:\Users\N0 GO BE\Downloads\JADE-all-4.5.0\JADE-bin-4.5.0>java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform1
Aug 07, 2022 1:07:30 PM jade.core.Runtime beginContainer
INFO: -----
This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
  
```

- Add a peripheral container to Platform1 by typing this in a new command line:
java -cp jade\lib\jade.jar jade.Boot -container

```

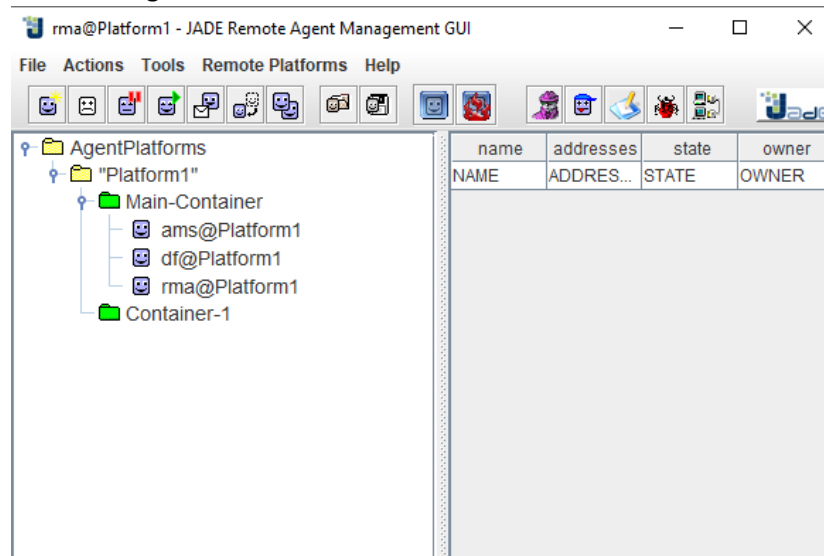
Command Prompt - java -cp jade\lib\jade.jar jade.Boot -container
(c) Microsoft Corporation. All rights reserved.

C:\Users\NO GO BE>cd Downloads\JADE-all-4.5.0\JADE-bin-4.5.0

C:\Users\NO GO BE\Downloads\JADE-all-4.5.0\JADE-bin-4.5.0>java -cp jade\lib\jade.jar jade.Boot -container
Aug 07, 2022 1:16:36 PM jade.core.Runtime beginContainer
INFO: -----
This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/

```

- You will have something similar to this:



- Start the second platform Platform2 by typing in a new command line:
java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform2 -local-port 1111

```

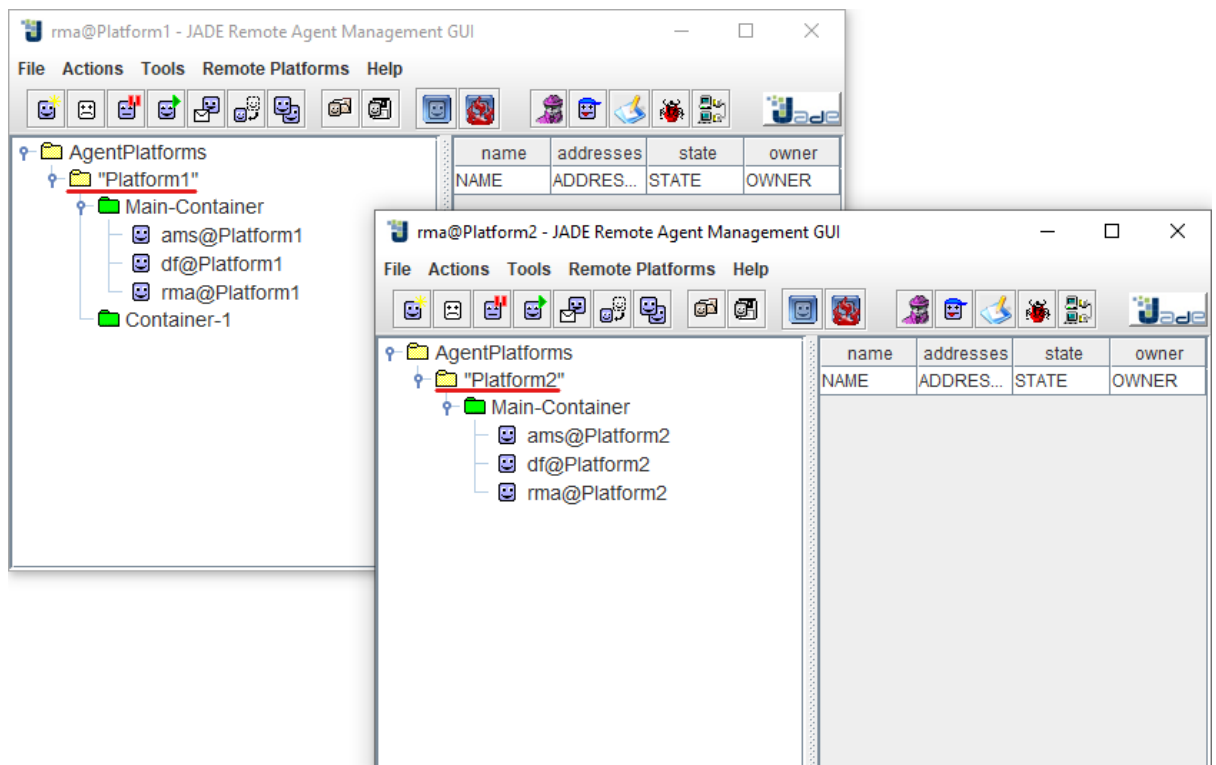
Command Prompt - java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform2 -local-port 1111
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\NO GO BE>cd Downloads\JADE-all-4.5.0\JADE-bin-4.5.0

C:\Users\NO GO BE\Downloads\JADE-all-4.5.0\JADE-bin-4.5.0>java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform2 -local-port 1111
Aug 07, 2022 1:21:05 PM jade.core.Runtime beginContainer
INFO: -----
This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/

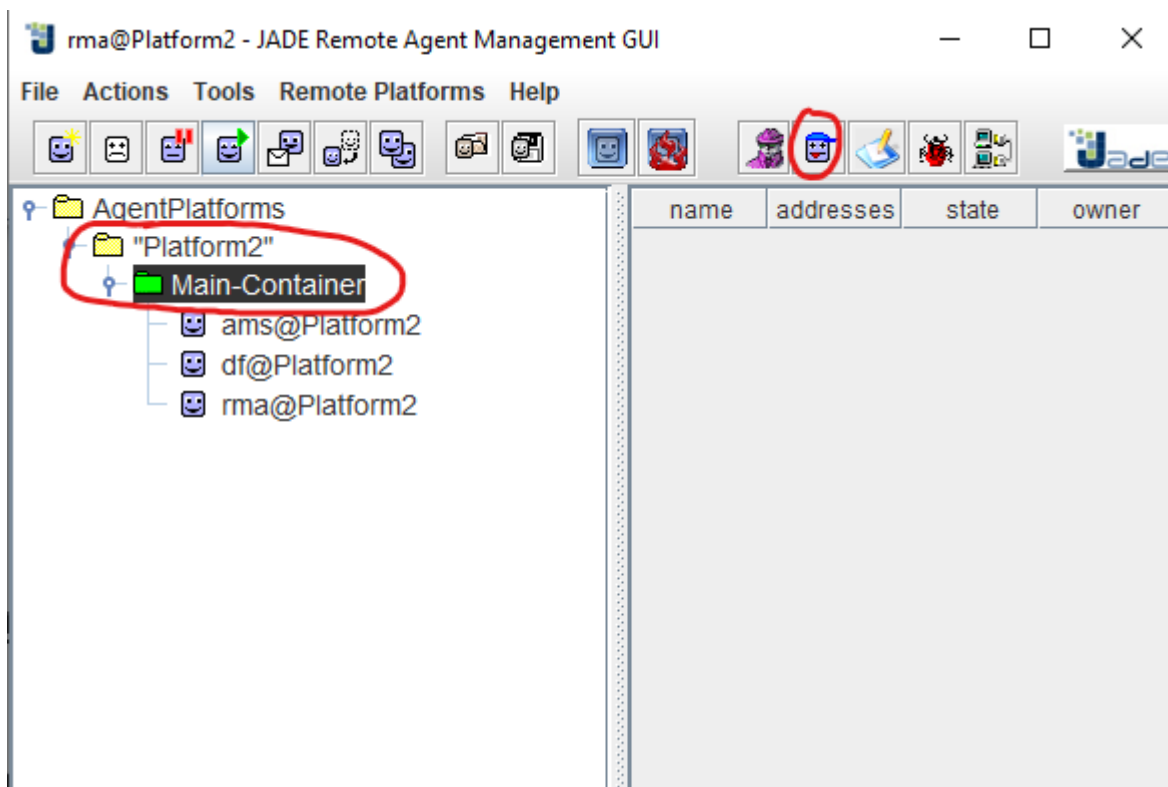
```

- You will have something like this:

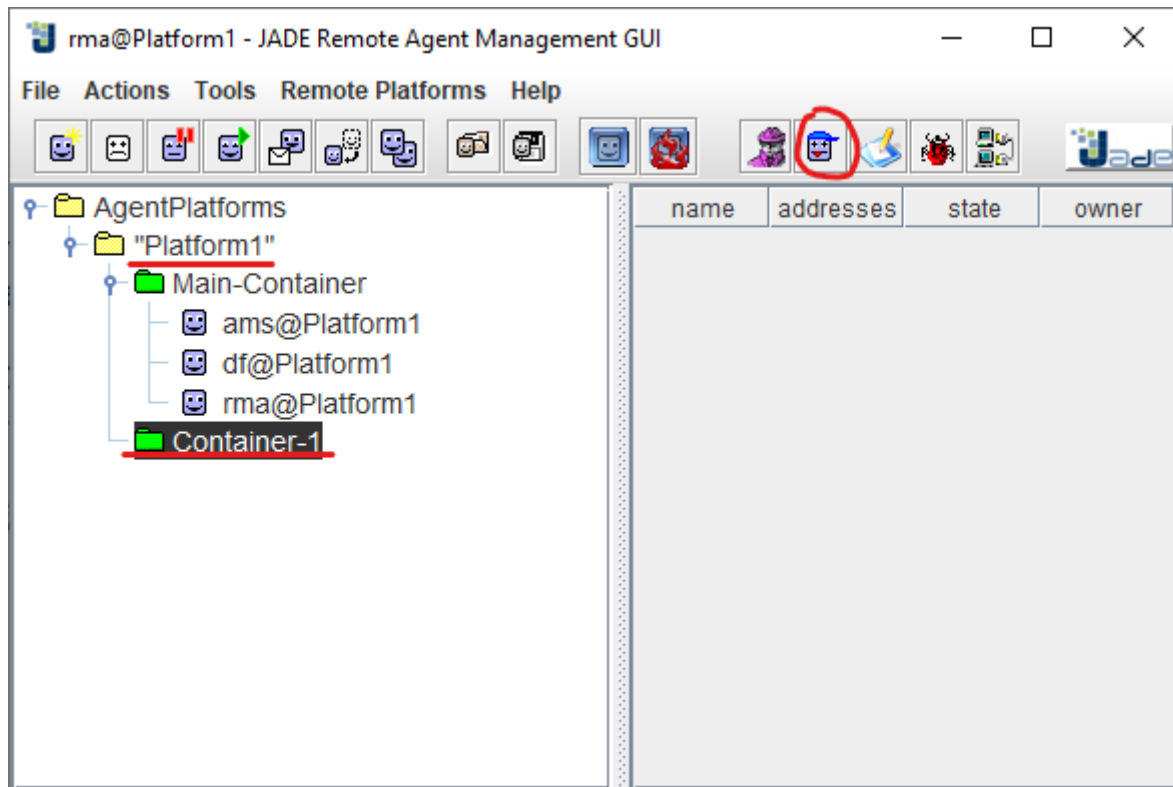


Inter-platform communication:

In this part, we will make a DummyAgent in Container-1/Platform1 to send a message to another DummyAgent in Platform2. To do it, firstly create a DummyAgent in Platform2 by clicking on the "Start DummyAgent" button from the Platform2 GUI:

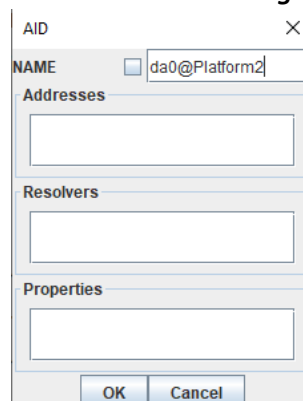


Now start a DummyAgent in Platform1/Container-1:



In Platform1 DummyAgent GUI, we set:

- Communicative act: INFORM
- Content: "Hello!"
- Right-click on the Receivers field to add a receiver's name. Type "da0@Platform2" for the name. **Do not check the local name check box! This is a global name.**



- Right-click on the Addresses field to add address of the receiver. Insert the receiver's address + "/acc". You can the address from the output of Platform2 Main Container:


```
Command Prompt - java -cp jade\lib\jade.jar jade.Boot -gui -platform-id Platform2 -local-port 1111

This is JADE 4.5.0 - revision 6825 of 23-05-2017 10:06:04
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/

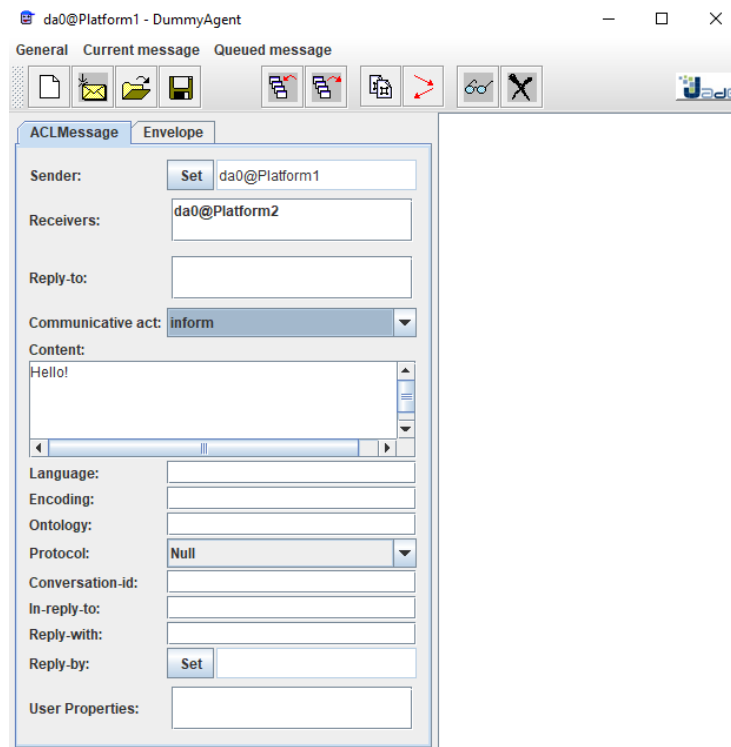
-----
Aug 07, 2022 7:03:35 PM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://192.168.20.98:1111

Aug 07, 2022 7:03:35 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Aug 07, 2022 7:03:35 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Aug 07, 2022 7:03:35 PM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Aug 07, 2022 7:03:35 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Aug 07, 2022 7:03:35 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Aug 07, 2022 7:03:35 PM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Aug 07, 2022 7:03:35 PM jade.mtp.http.HTTPServer <init>
WARNING: Port 7778 is already in used, selected another one
Aug 07, 2022 7:03:35 PM jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://192.168.56.1:63222/acc
Aug 07, 2022 7:03:35 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@192.168.20.98 is ready.
-----
```

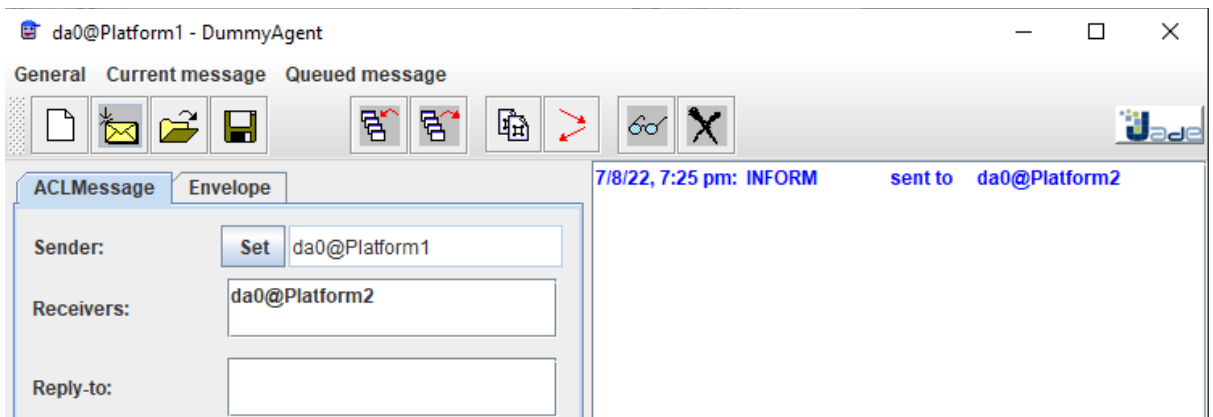
The image shows a dialog box titled "AID" with a close button (X) in the top right corner. It contains several sections for configuring an agent:

- NAME:** A text field containing "da0@Platform2".
- Addresses:** A text field containing "http://192.168.20.98:1111/acc".
- Resolvers:** An empty text field.
- Properties:** An empty text field.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

- This is what you have after doing the above configurations:



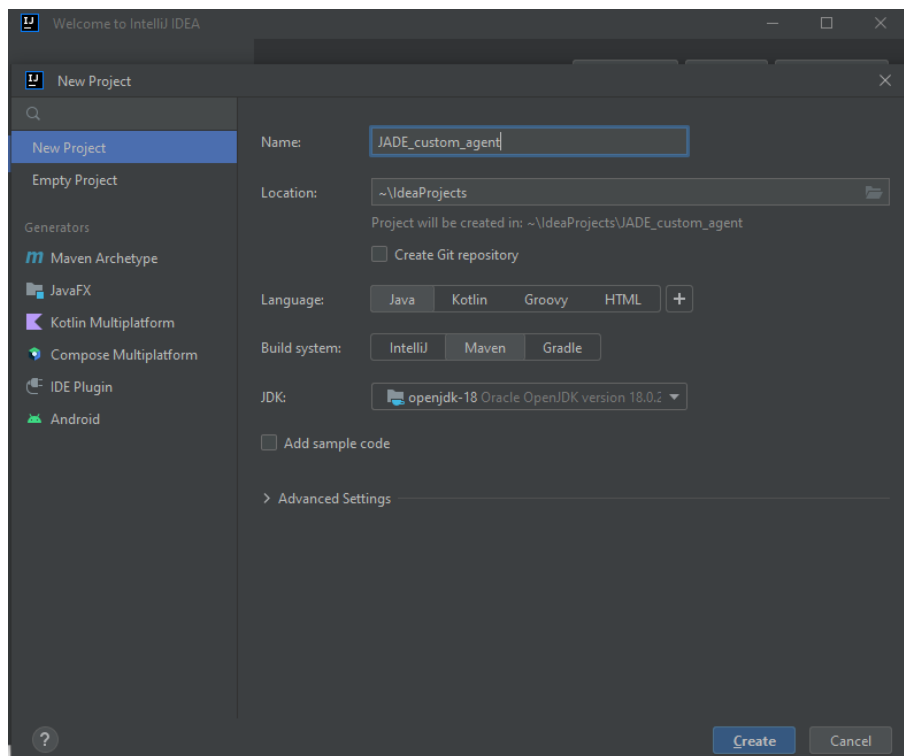
- Now try to send the message, you will see a message confirms that the INFORM message has been sent to Platform2:



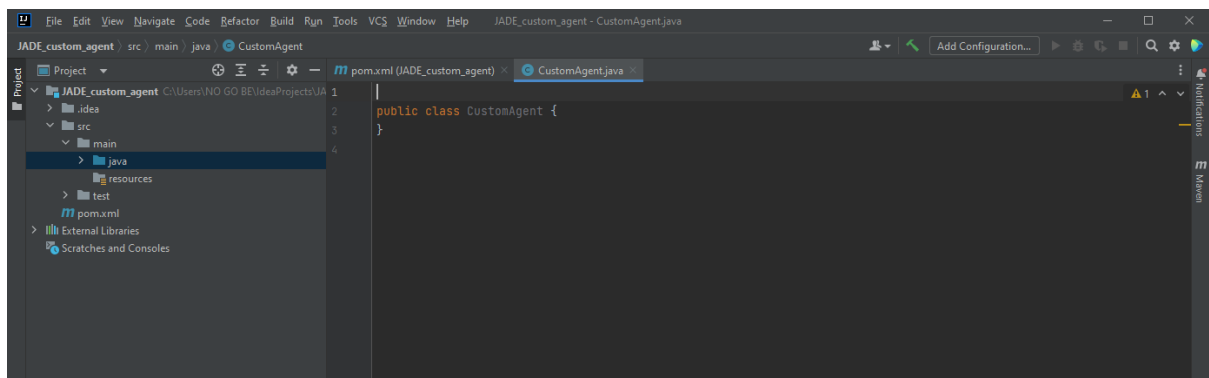
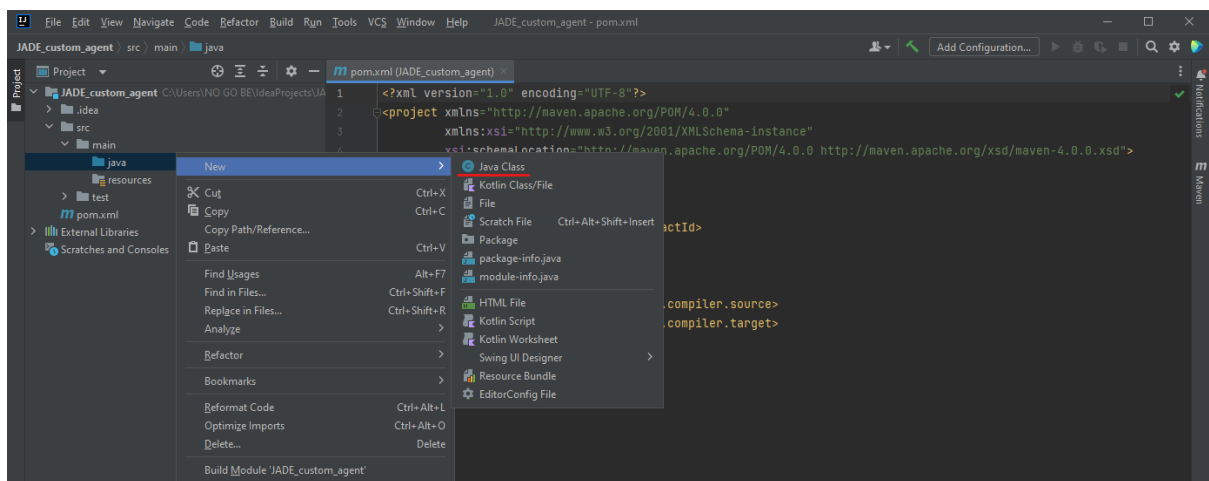
4. Create JADE custom agent

Last week, we have seen how to call and run example agents that are included in JADE when we download it. In this part, we will learn how to create our custom agent.

Firstly, you need to add JADE framework to IntelliJ (or any other IDEs that you are using). Let's create a new project name "JADE_custom_agent":



Create a new Java class name “CustomAgent” to write our custom agent:

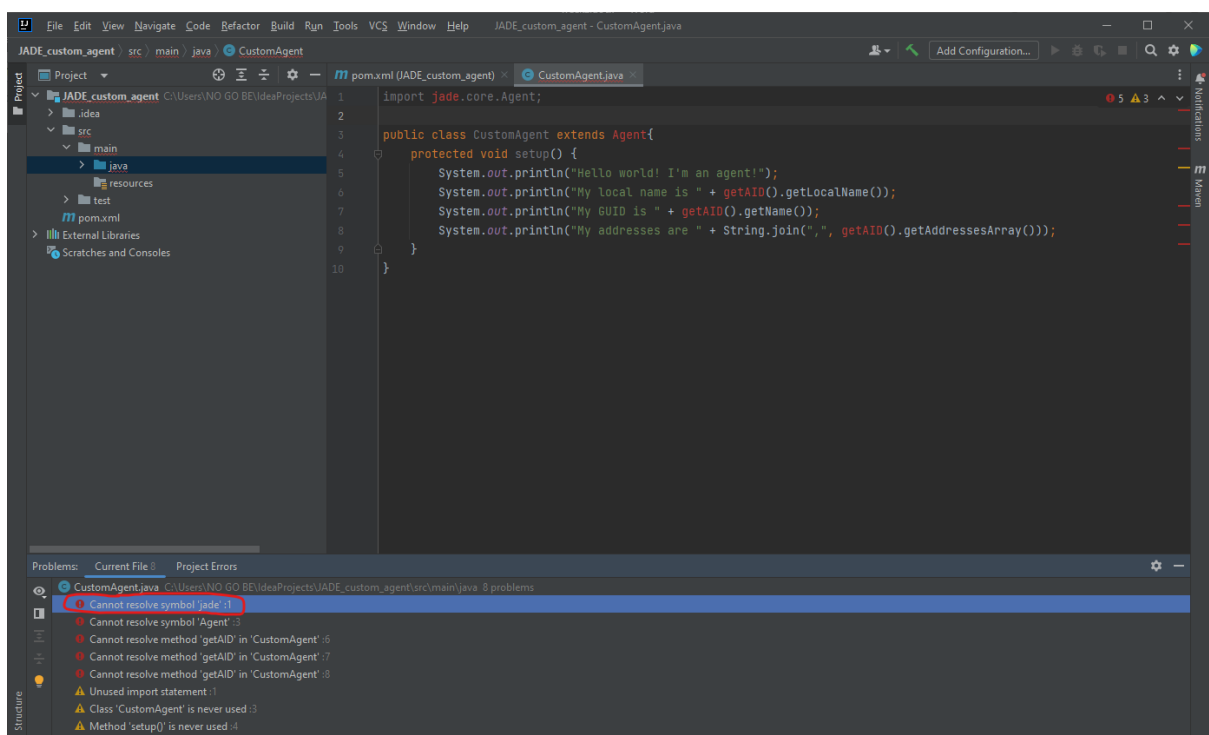


Copy the following code to configure our new custom agent:

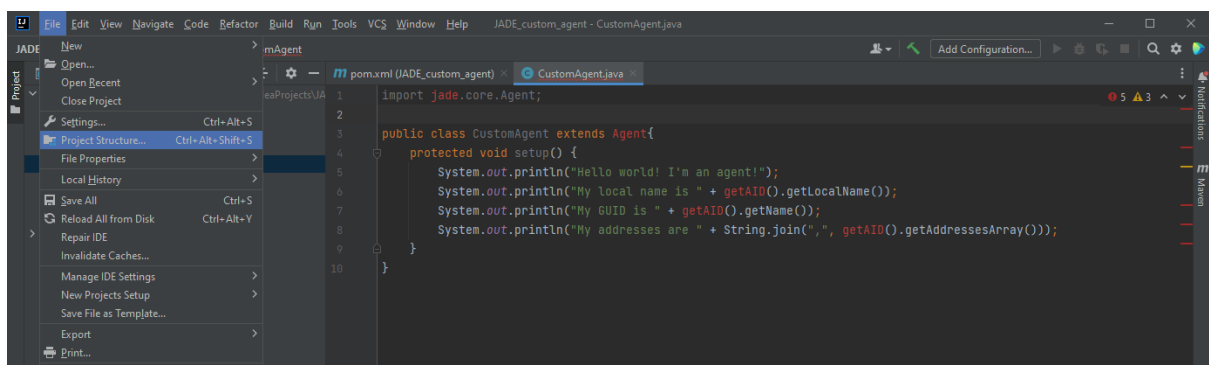
```
import jade.core.Agent;

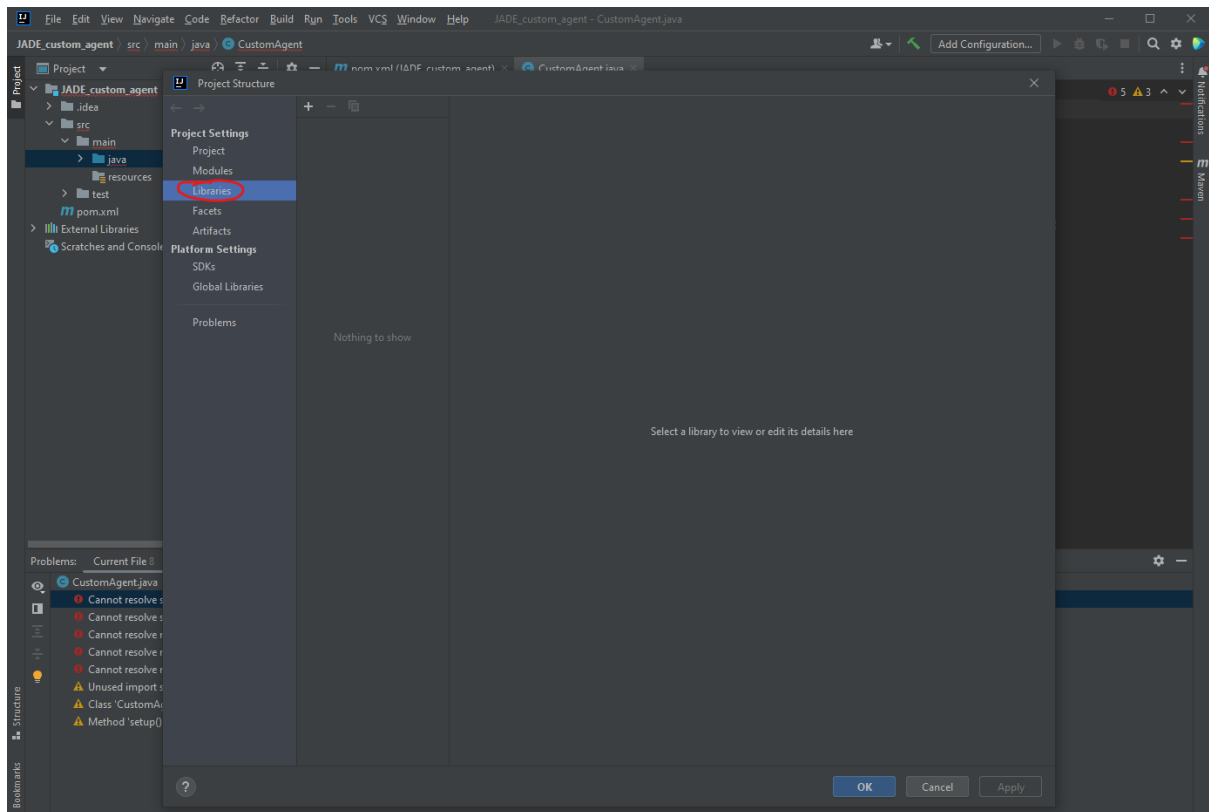
public class CustomAgent extends Agent{
    protected void setup() {
        System.out.println("Hello world! I'm an agent!");
        System.out.println("My local name is " + getAID().getLocalName());
        System.out.println("My GUID is " + getAID().getName());
        System.out.println("My addresses are " + String.join(", ",
getAID().getAddressesArray()));
    }
}
```

You will see some errors because we haven't add the JADE library in this environment, so it cannot detect `jade.core.Agent` package:

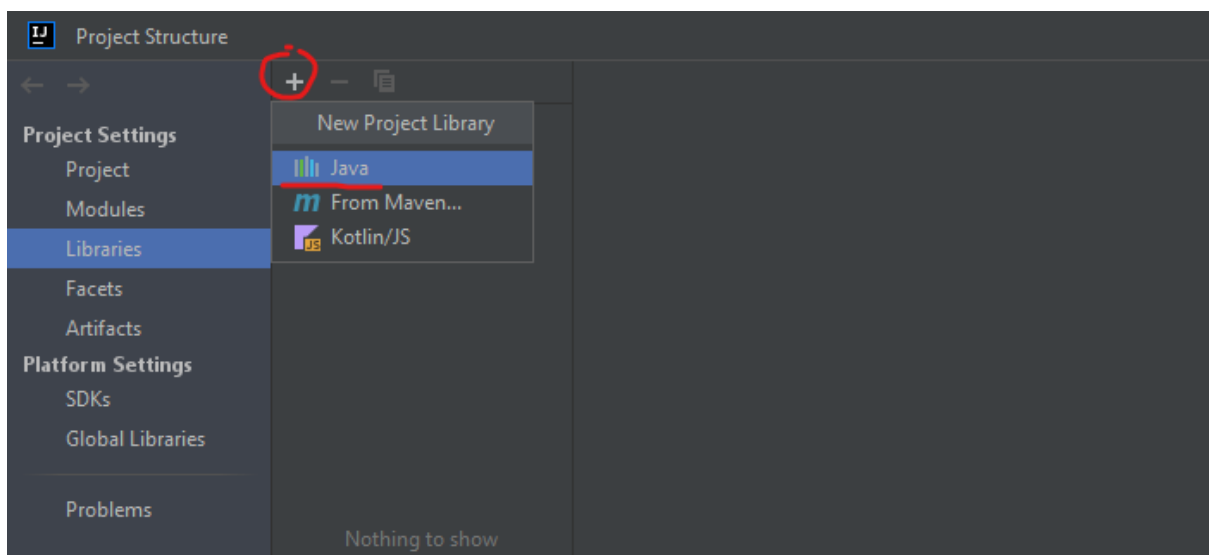


To add JADE, go to File -> Project Structure, and choose “Libraries” under the “Project Settings” to create a new folder for your JADE libraries:

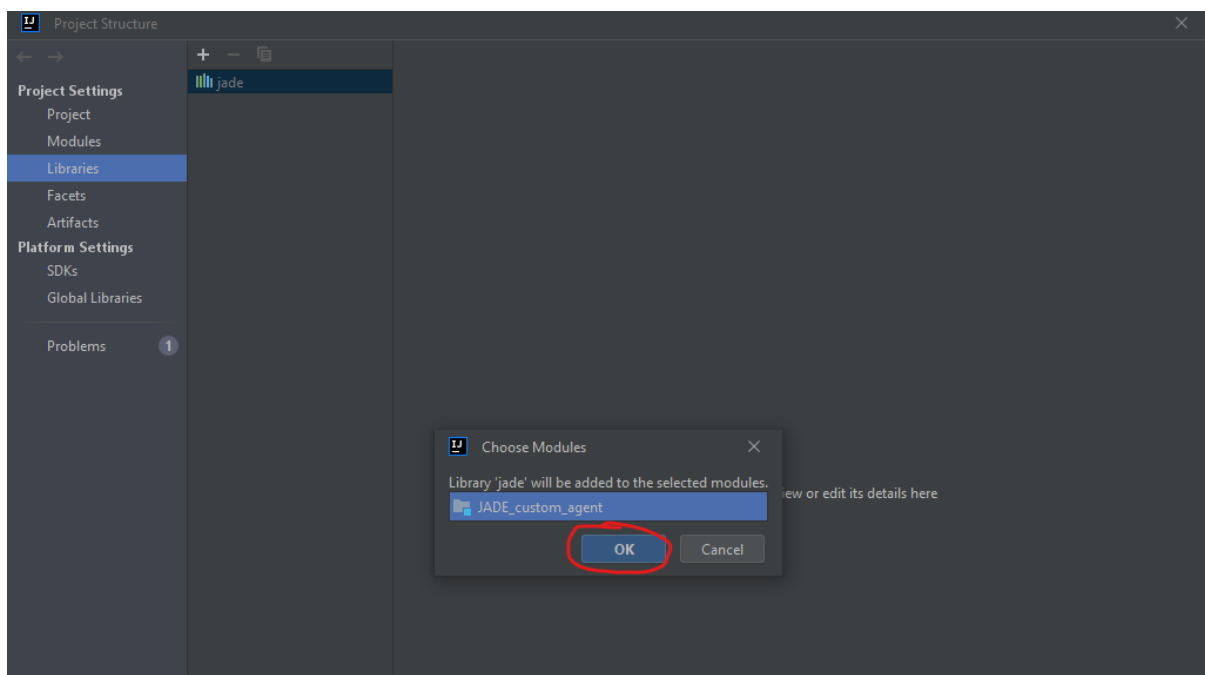
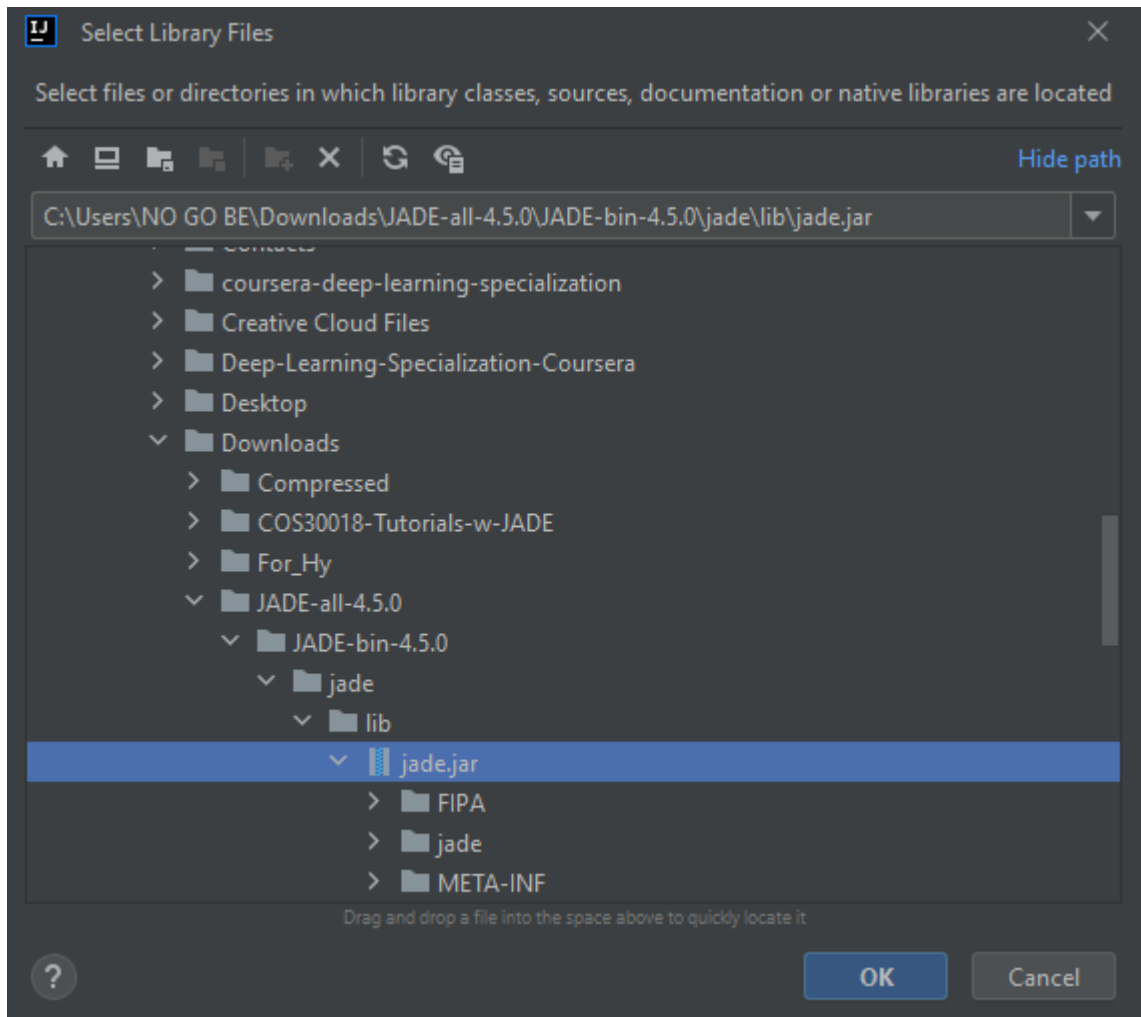




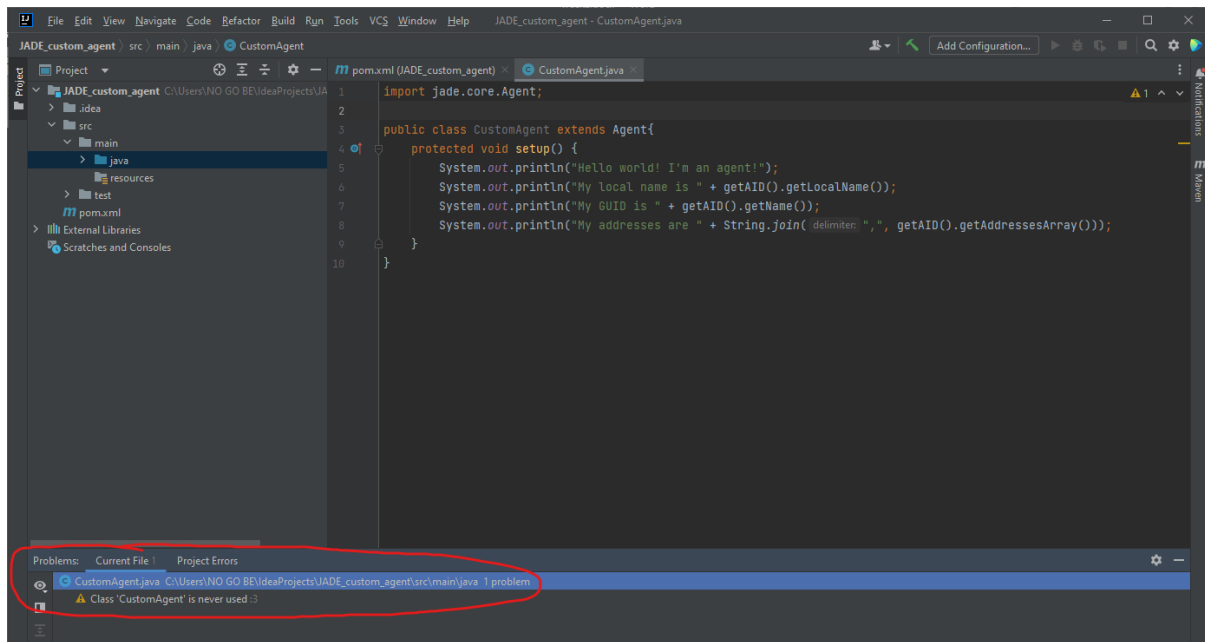
Click on the “+” button and choose “Java” to add a Java library:



Then navigate to the location where you keep your “**jade.jar**” file and add it:



Click Apply and close the Project Structure window. You can see the errors are gone now:



That's it! Creating a JADE agent is as simple as defining a class extending the `jade.core.Agent` class and implementing the `setup()` method as shown in the code below:

Agent identifiers:

Each agent is identified by an “agent identifier” represented as an instance of the `jade.core.AID` class. The `getAID()` method of the `Agent` class allows retrieving the agent identifier. An AID object includes a globally unique name plus a number of addresses. The name in JADE has the form `<nickname>@<platform-name>` so that an agent called *Peter* living on a platform called *P1* will have *Peter@P1* as globally unique name. The addresses included in the AID are the addresses of the platform the agent lives in. These addresses are only used when an agent needs to communicate with another agent living on a different platform.

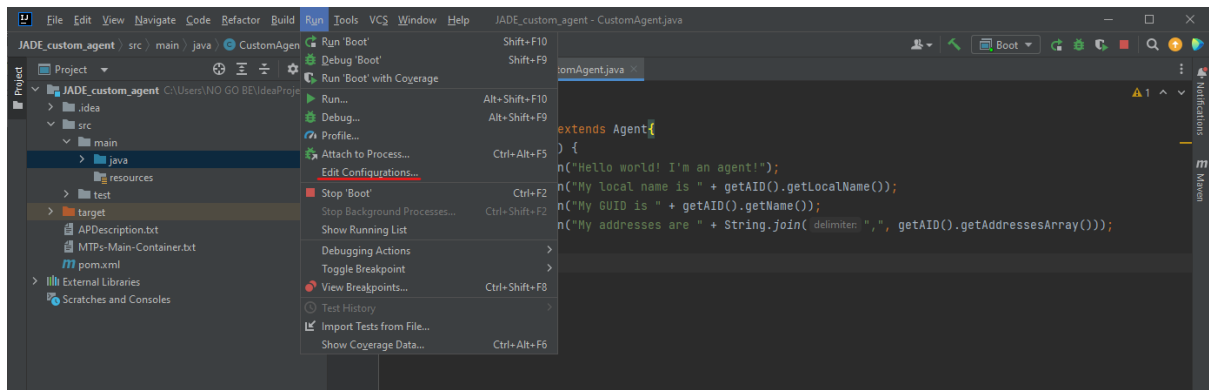
Knowing the nickname of an agent, its AID can be obtained as follows:

```
String nickname = "Peter";
AID id = new AID(nickname, AID.ISLOCALNAME);
```

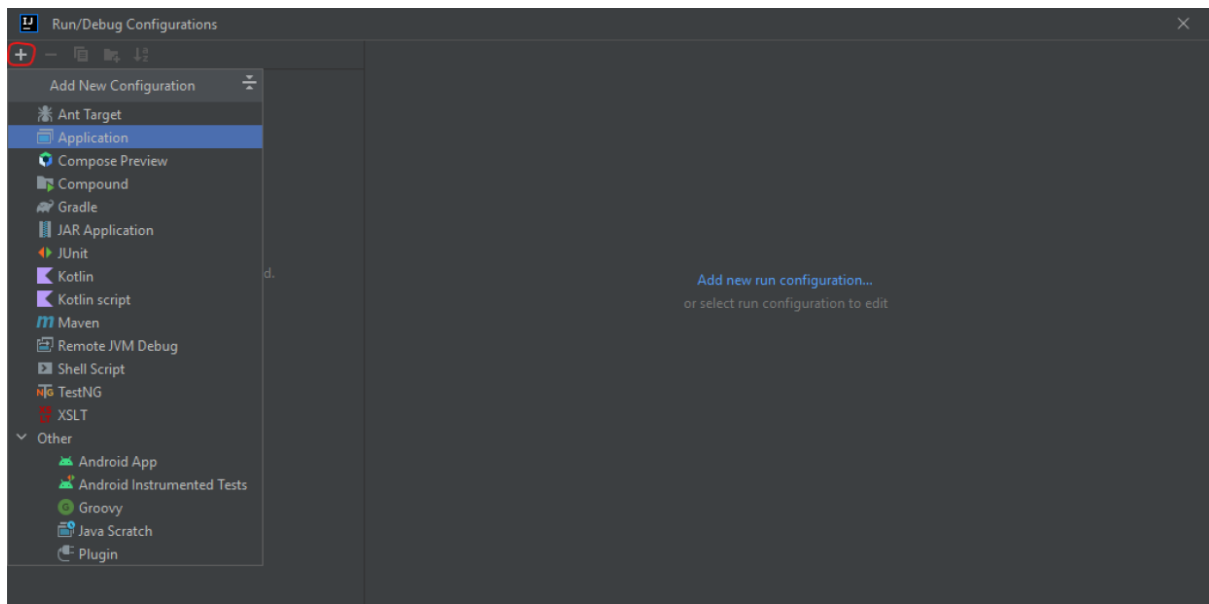
The `ISLOCALNAME` constant indicates that the first parameter represents the nickname (local to the platform) and not the globally unique name of the agent.

Running agents:

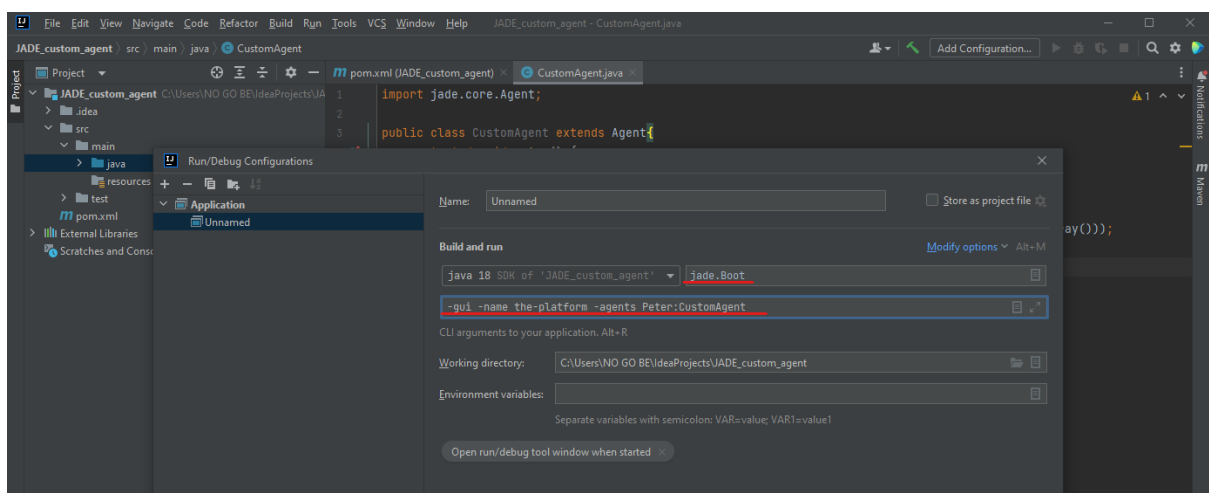
Go to the Edit Configurations option under the Run menu:



Click on “Add New Configuration” and choose “Application”:



In the popup menu, you type “jade.Boot” for the Main class field and “-gui -name the-platform -agents Peter:CustomAgent” for the Program arguments field:



This names our agent platform ‘the-platform’ and creates an agent called ‘Peter’ which is an instance of the class ‘CustomAgent’ we wrote before.

Click Apply and close the Configurations window.

Hit Run to start your JADE runtime and execute your custom agent. You will see a RMA GUI showed up and the custom agent prints out the scripts on the console:

