OBJECTIVE: Programing agents such that one sends a number and the receiver squares the value and sends back the number.

Project structure:

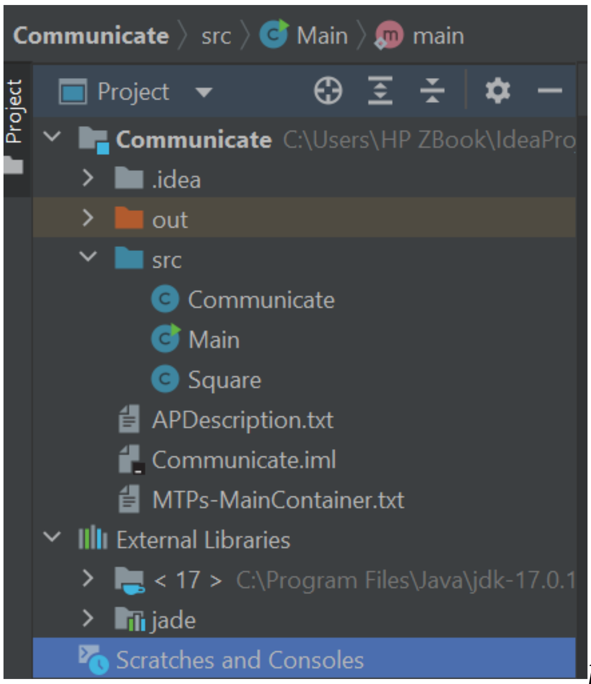
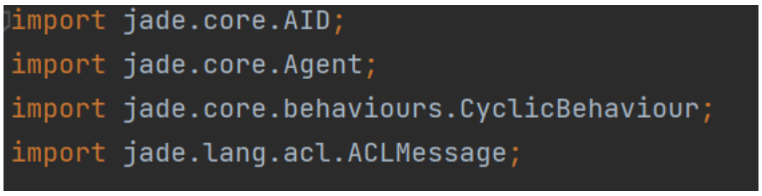


fig: project structure

I created 3 classes:

Communicate.java - scans a number from the keyboard and sends it to the agent in class Square.java that will square it.

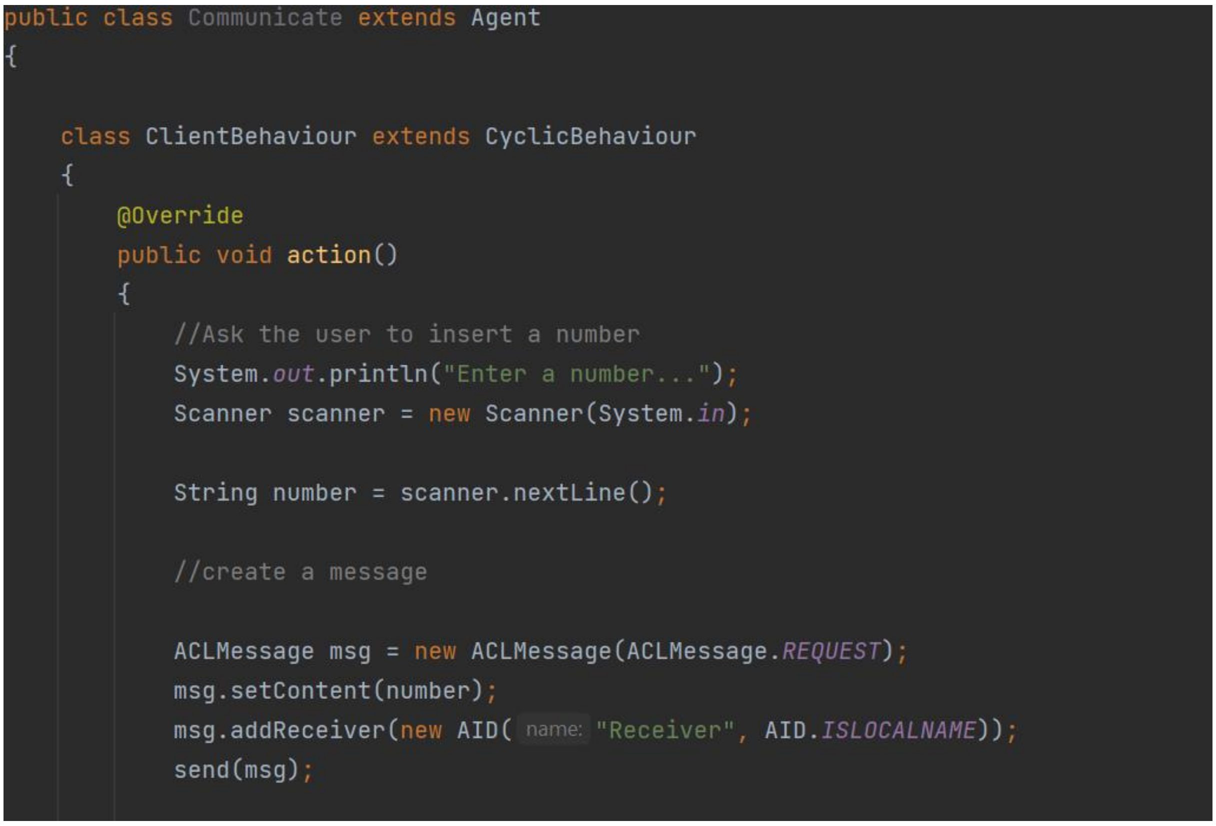
The main class is Main.java.



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

fig: The imports

Communicate class



```
public class Communicate extends Agent
{

    class ClientBehaviour extends CyclicBehaviour
    {
        @Override
        public void action()
        {
            //Ask the user to insert a number
            System.out.println("Enter a number...");
            Scanner scanner = new Scanner(System.in);

            String number = scanner.nextLine();

            //create a message

            ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
            msg.setContent(number);
            msg.addReceiver(new AID( name: "Receiver", AID.ISLOCALNAME));
            send(msg);
```

fig: Communicate class

I created a class of Client's behavior that extends *Cyclic behavior*.

Cyclic behavior is an abstract class that can be extended by application programmers to create behaviors that keep executing continuously. This will allow the client to continuously scan for values and be sending messages to the receiver. It stays active as long as its agent is alive.

Scanner

The *Scanner* class is used to get user input, and it is found in the *java*.util package.[1]

It used for obtaining the input of the primitive types like int, double, etc. and strings.

```
Scanner scanner = new Scanner(System.in);
```

```
String number = scanner.nextLine();
```

Creating the message and its contents:

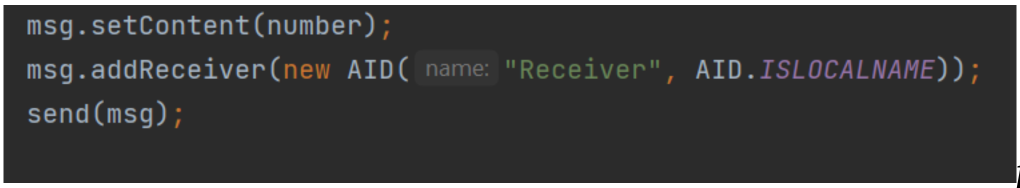
Jade follows FIPA standards so that ideally Jade agents could interact with agents written in other languages and running on other platforms.

In JADE, messages adhere strictly to the ACL (Agent Communication Language) standard which allows several possibilities for the encoding of the actual *content*.

```
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
```

I created an instance of *ACLMessage* with the informative *REQUEST*.

REQUEST allows to ask the other agent to do something [3].



```
msg.setContent(number);
msg.addReceiver(new AID( name: "Receiver", AID.ISLOCALNAME));
send(msg);
```

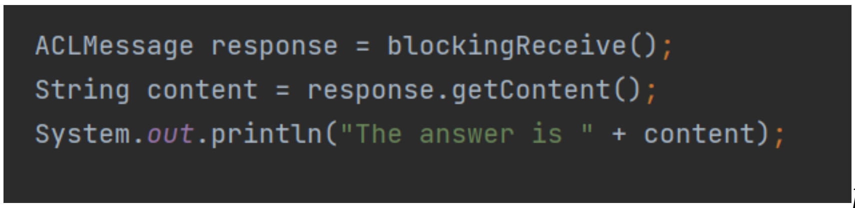
fig: msg

Content - This is the main content of the message. And in our case, it’s the scanned number.

We use ***add****Receiver* because there is no ***set****Receiver* method. One reason is that we can add several receivers to the message and the one **send** broadcasts it to all of them. [3]

Messages are routed to wherever the destination agent resides and are placed in its message queue.

Receiving and printing Output

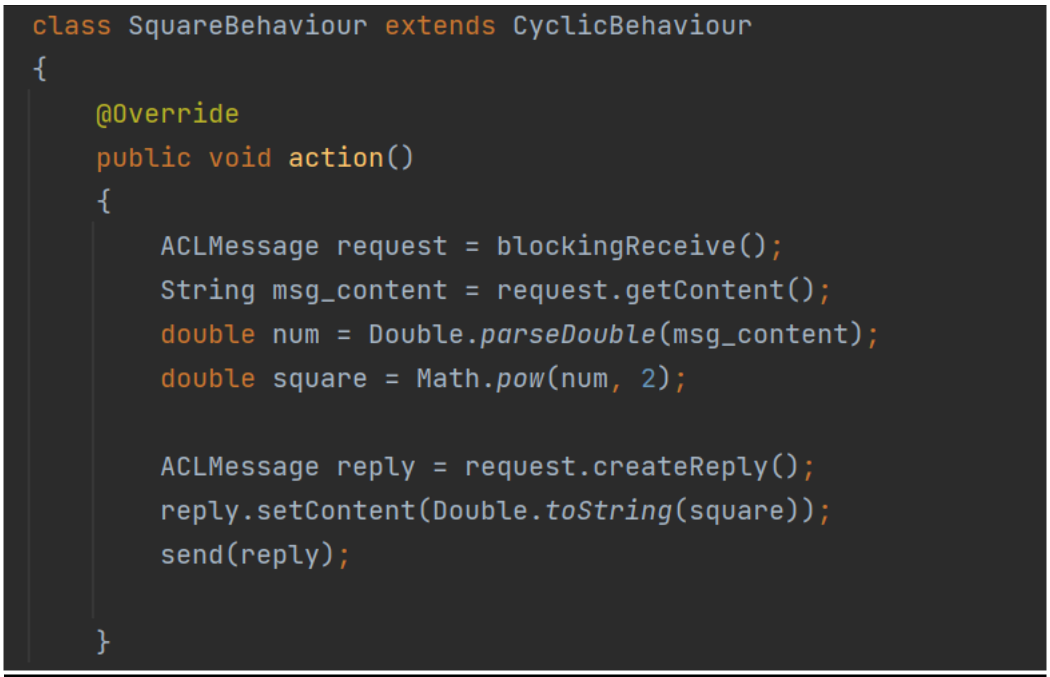


```
ACLMessage response = blockingReceive();
String content = response.getContent();
System.out.println("The answer is " + content);
```

fig: receiving

There are 2 basic ways for the receiver to get its messages. By using *blockingReceive()*, as used in the project, the receiving agent suspends all its activities until a message arrives.

The Square class



```
class SquareBehaviour extends CyclicBehaviour
{
    @Override
    public void action()
    {
        ACLMessage request = blockingReceive();
        String msg_content = request.getContent();
        double num = Double.parseDouble(msg_content);
        double square = Math.pow(num, 2);

        ACLMessage reply = request.createReply();
        reply.setContent(Double.toString(square));
        send(reply);

    }
```

This class also has a cyclic behavior to allow it to continue receiving messages and sending them back.

This agent will receive the content sent by the first agent. And in receiving it using *blockingReceive()*, it will suspend all its activities.

Since *setContent* method accepts string as argument, to square the number, we have to convert it to a double value first.

This can be done by using the Double.ParseDouble() method.

```
ouble square = Math.pow(num, 2);
```

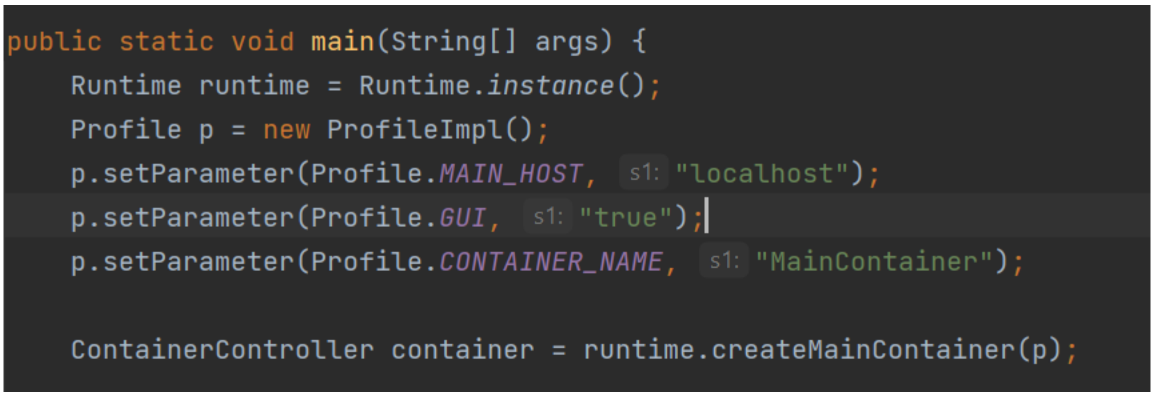
The Math class's pow method returns the value of the first argument raised to the power of the second argument which is 2 in our case.

Jade provides a method *createReply()* which creates a new message with the sender and receiver attributes switched and all other attributes set correctly. Generally, only the content and performative have to be modified before sending it back.

```
reply.setContent(Double.toString(square));
```

In the same way, we convert the squared number to a String value which can be taken in by the *SetContent*. And finally send the reply back to the sender.

The main class



```
public static void main(String[] args) {
    Runtime runtime = Runtime.instance();
    Profile p = new ProfileImpl();
    p.setParameter(Profile.MAIN_HOST, s1: "localhost");
    p.setParameter(Profile.GUI, s1: "true");
    p.setParameter(Profile.CONTAINER_NAME, s1: "MainContainer");

    ContainerController container = runtime.createMainContainer(p);
```

fig:Main

The runtime instance

We create an instance of runtime. This is so that it knows how to execute our code provided by jade. Instanciating of this class that should be then used to create agent containers.

The profile class allows retrieving configuration-dependant classes.

This class, **ProfileImpl,** extends the Profile class. It allows the JADE core to retrieve configuration-dependent classes and boot parameters.
The line of code :

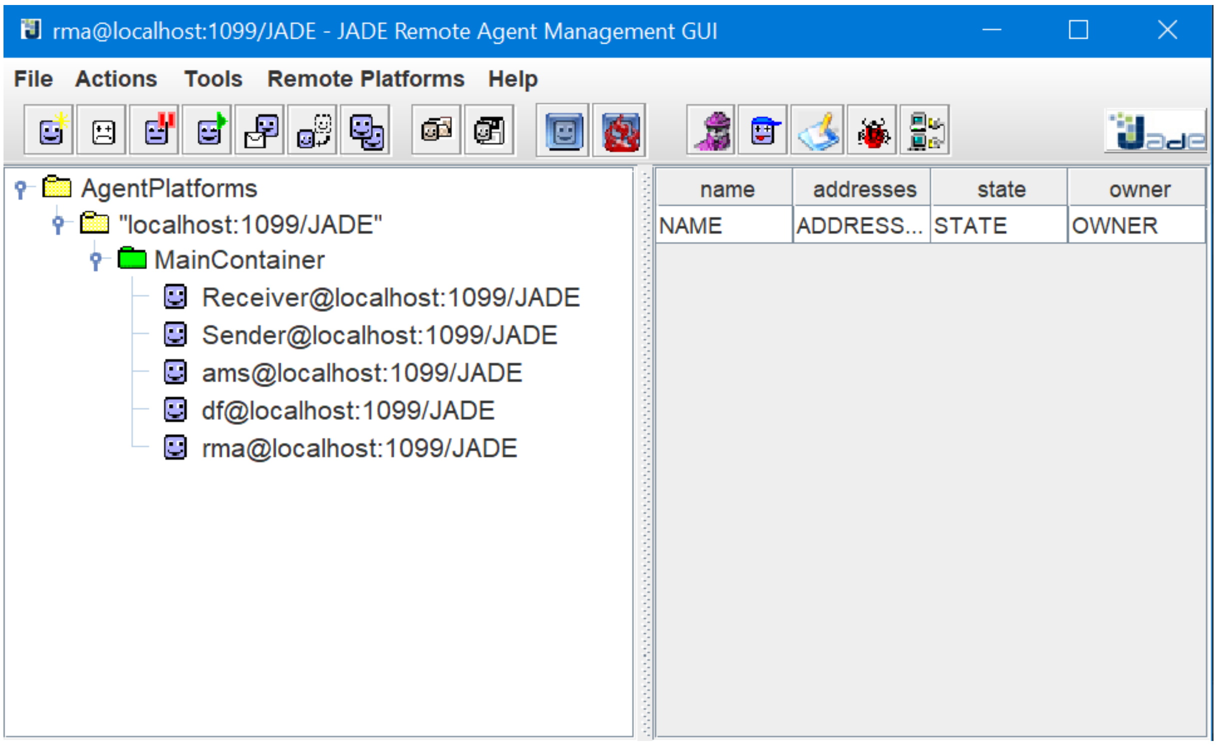
```
p.setParameter(Profile.CONTAINER_NAME, "MainContainer");
```

Allows to put both agents in the main container.

CreateNewAgent creates a new JADE agent, running within this container. With parameters:

- *Name* – the unique name for the newly created agent (in our program its an integer value).
- *className* – which is name of the class that implements the agent
- *Arguments* – an object array with initialization parameters

The start method will call the setup method for the agent.


rma@localhost:1099/JADE - JADE Remote Agent Management GUI

File Actions Tools Remote Platforms Help

- AgentPlatforms
  - "localhost:1099/JADE"
    - MainContainer
      - Receiver@localhost:1099/JADE
      - Sender@localhost:1099/JADE
      - ams@localhost:1099/JADE
      - df@localhost:1099/JADE
      - rma@localhost:1099/JADE

name	addresses	state	owner
NAME	ADDRESS...	STATE	OWNER

Fig: container

<u>References</u>

1. https://www.google.com date visited 3 May 2022
2. https://www.programcreek.com/java-api-examples/?class=jade.lang.acl.ACLMessage&method=REQUEST date visited 5 May 2022
3. https://www.iro.umontreal.ca/~vaucher/Agents/Jade/primer4.html date visited 5 May 2022