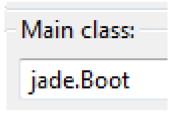
JADE in JAVA – rapid tutorial

Mateusz Kaflowski AGH WIMIiP 2013

mkaflowski@gmail.com

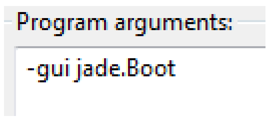
1. How to run JADE in Eclipse GUI?

In „Run Configurations”:



Main class:

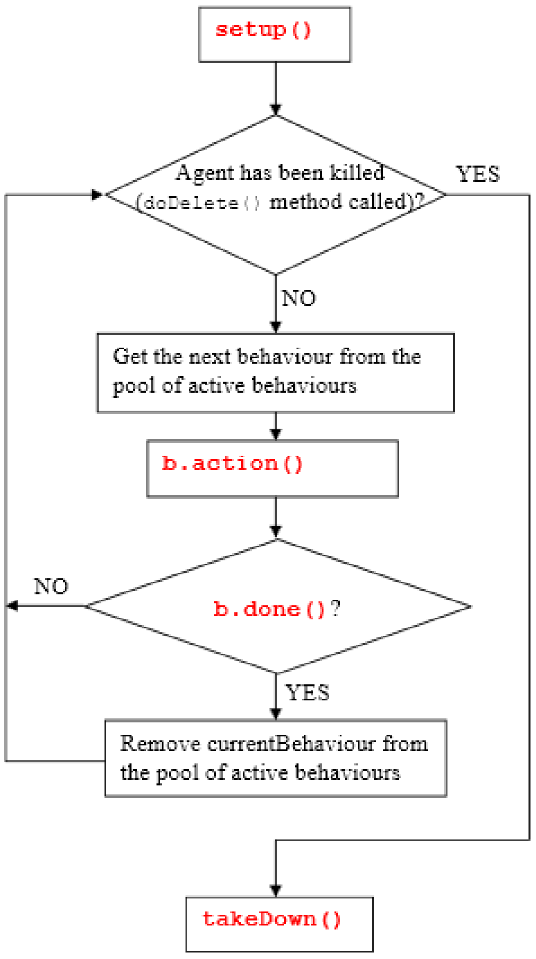
jade.Boot



Program arguments:

-gui jade.Boot

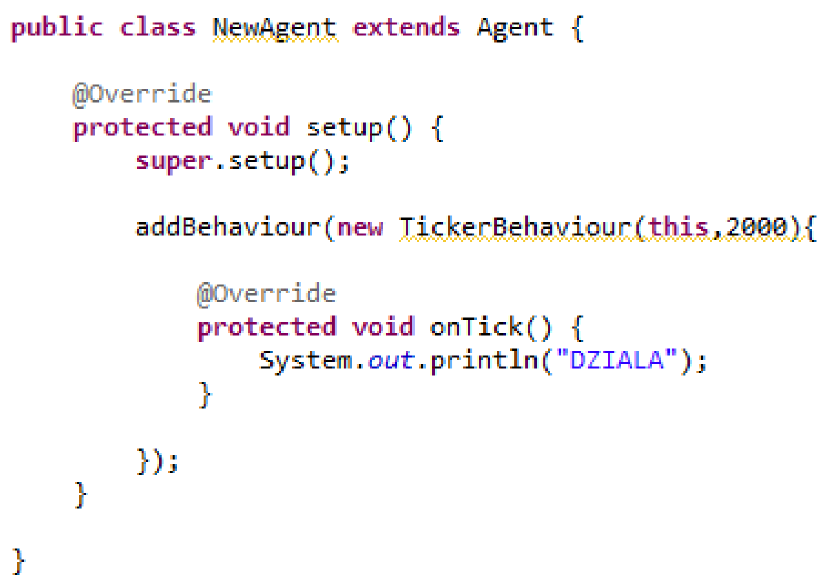
2. What is aget lifecycle:


setup()
Agent has been killed (doDelete() method called)?
YES
NO
Get the next behaviour from the pool of active behaviours
b.action()
NO
b.done()?
YES
Remove currentBehaviour from the pool of active behaviours
takeDown()

3. What kind of behaviours(actions) can have agent?

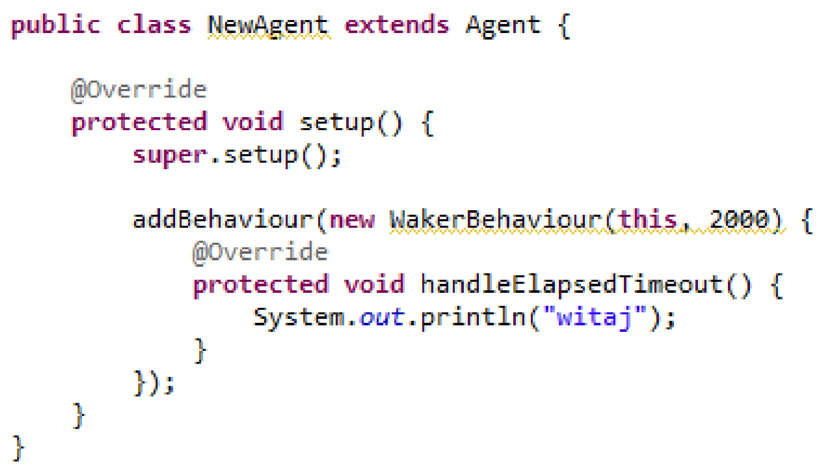
- OneShotBehaviour – single action
- CyclickBehaviour – actions in loop
- WakerBehaviour – single action after waiting some time
- TickerBehaviour – actions in loop with pauses

4. Code agent who types “DZIALA” in loop with 2 second pauses.



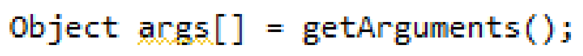
```java
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new TickerBehaviour(this,2000){

            @Override
            protected void onTick() {
                System.out.println("DZIALA");
            }

        });
    }

}
```

5. Code agent who types “witaj” after 2 seconds.



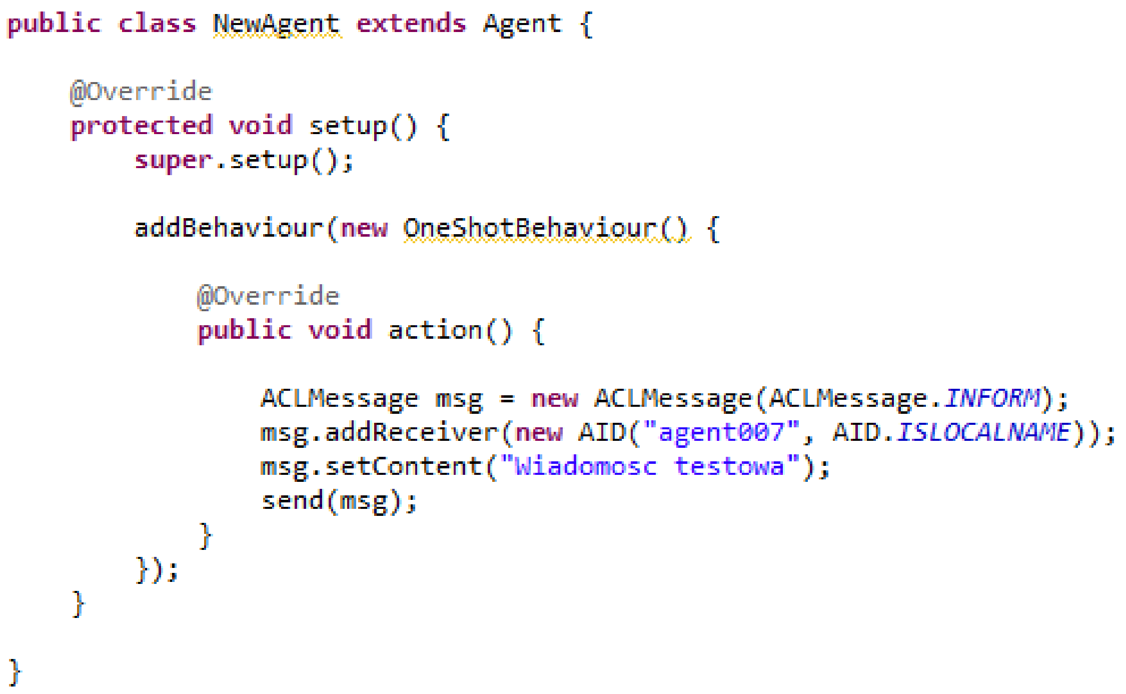
```java
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new WakerBehaviour(this, 2000) {
            @Override
            protected void handleElapsedTimeout() {
                System.out.println("witaj");
            }
        });
    }
}
```

6. How to get program arguments in agent code?



```
Object args[] = getArguments();
```

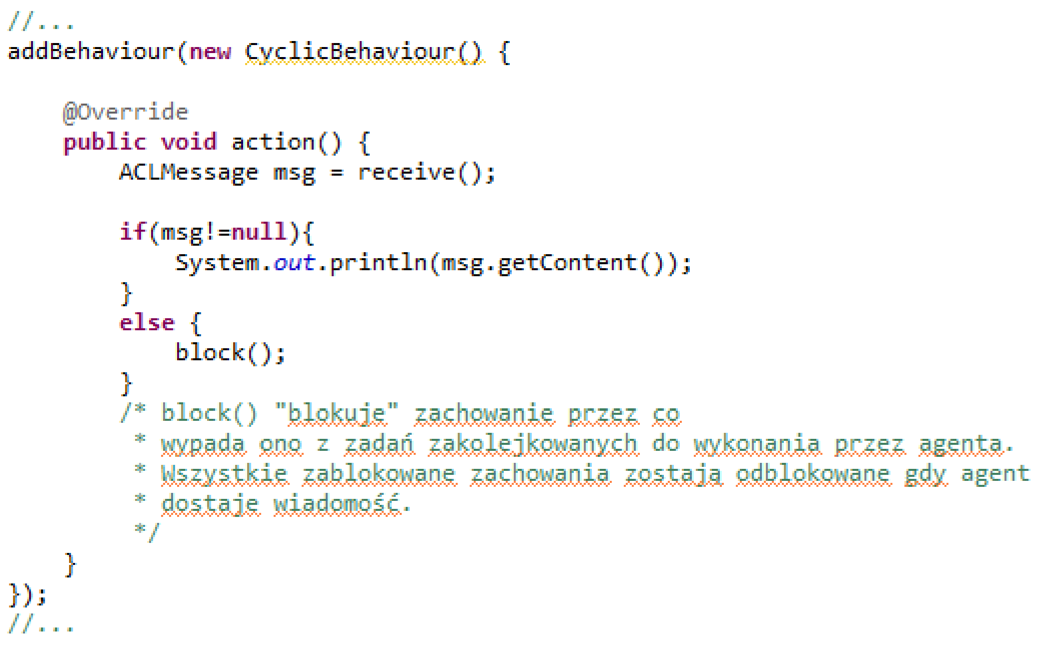
7. Code agent who sends message to another agent.



```
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new OneShotBehaviour() {

            @Override
            public void action() {

                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(new AID("agent007", AID.ISLOCALNAME));
                msg.setContent("Wiadomosc testowa");
                send(msg);
            }
        });
    }

}
```

The message will be send to agent who local name is „agent007".
If you want to send objects use `msg.setContentObject(obj);`

8. Receive and type message.



```
//...
addBehaviour(new CyclicBehaviour() {

    @Override
    public void action() {
        ACLMessage msg = receive();

        if(msg!=null){
            System.out.println(msg.getContent());
        }
        else {
            block();
        }
        /* block() "blokuje" zachowanie przez co
         * wypada ono z zadań zakolejkowanych do wykonania przez agenta.
         * Wszystkie zablokowane zachowania zostają odblokowane gdy agent
         * dostaje wiadomość.
         */
    }
});
//...
```

9. Register yourself (agent) in Yellow Pages.

```
//...
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());

ServiceDescription sd = new ServiceDescription();
sd.setName("Nazwa_serwisu");
sd.setType("Typ_serwisu");

dfd.addServices(sd);

try {
    DFService.register(myAgent, dfd);
} catch (FIPAException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
//...
```

10. Where sou should deregiser from YP?

When agent is dying. Just override:

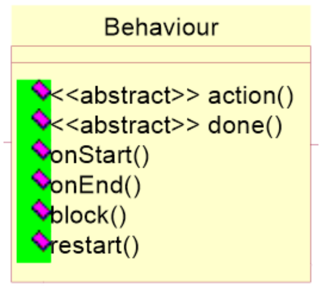
```
@Override
protected void takeDown() {
    super.takeDown();

    try {
        DFService.deregister(this);
    } catch (FIPAException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

11. Find and type agents who have registered service „Typ_serwisu”.

```
// ...
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("Typ_serwisu");

dfd.addServices(sd);

try {
    DFAgentDescription results[] = DFService.search(myAgent,
            dfd);
    for (int i = 0; i < results.length; i++) {
        System.out.println(results[i].getName().getLocalName());
    }
} catch (FIPAException e) {
    e.printStackTrace();
}
// ...
```

12. onStart(), onEnd() methods.

They are in Behaviour, and are the "prologue" and "epilogue". They are empty and can be overwritten in child classes. Before performing the action of the agent's behavior will be executed method onStart() (one time even for CyclicBehaviour). Similarly, on onEnd() except that it will do at the end of performed action. onEnd zwrana int'a - exit code of conduct.



13. How to clone and migrate agent?

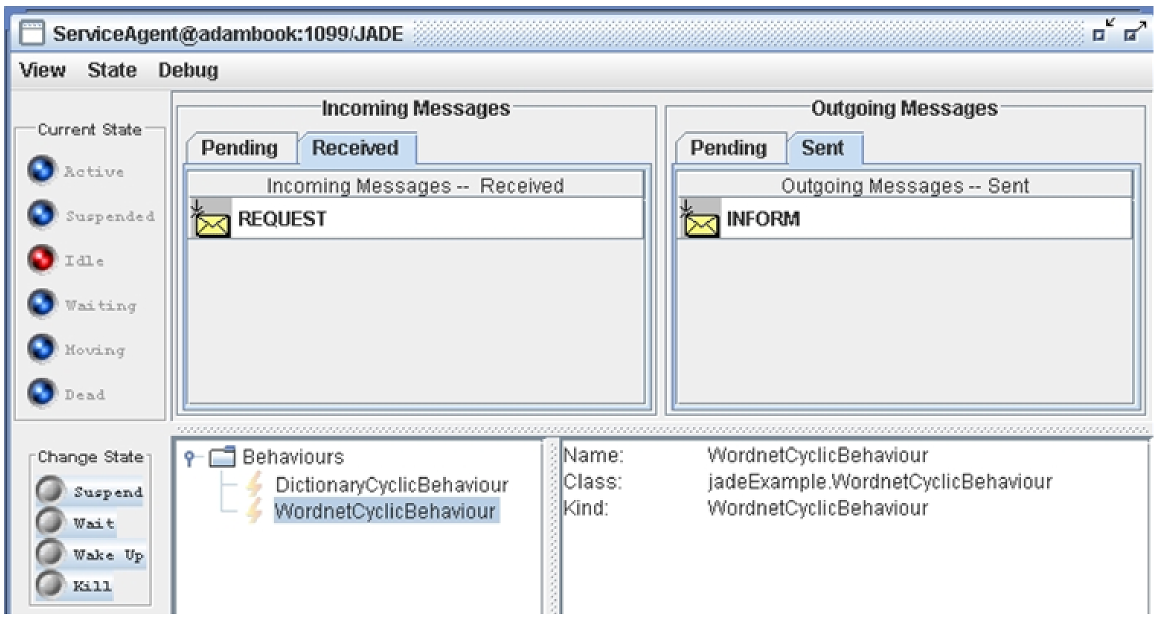
doClone(Location arg0, String arg1) and doMove(Location arg0).

14. Can agent shut down platform?

AMS (Agent Management System) is a special agent and only he has the ability to create and delete other agents, containers or closed platform. The agent may, however, require the closure of the AMS platform.

15. Inspector agent.

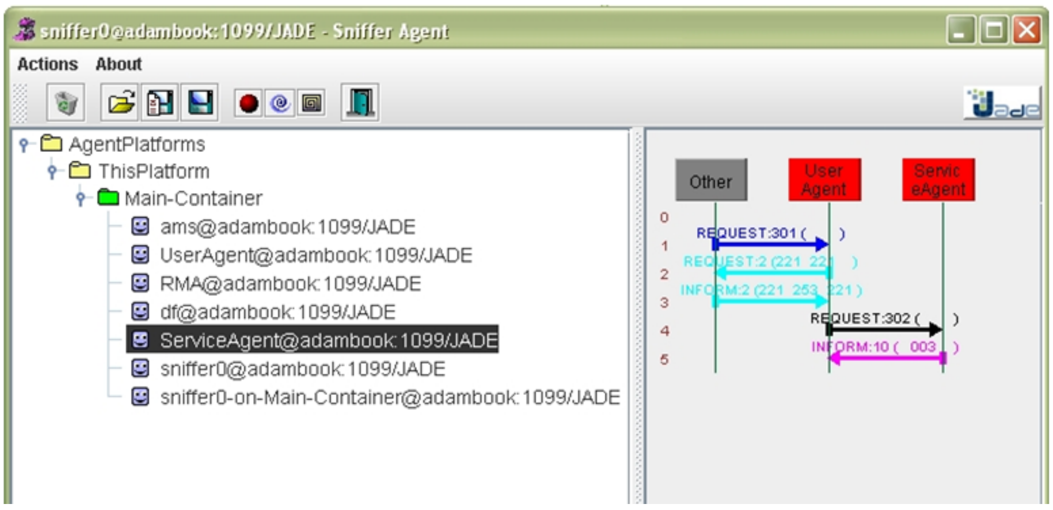
IntrospectorAgent allows you to select the agent and monitoring its behavior and messages that come to him and he is sent, it is possible to both view the agent message queue (waiting to receive) as well as those received.



Name: WordnetCyclicBehaviour
Class: jadeExample.WordnetCyclicBehaviour
Kind: WordnetCyclicBehaviour

16. Sniffer agent.

Sniffer shows the flow of messages between agents in the perspective of many agents that have been selected as objects monitoring. You can tell that he catches the messages on the fly, and shows us their use. Creates diagrams similar to UML. Sample record of the interaction between agents rma (gui), UserAgent and ServiceAgent as follows:

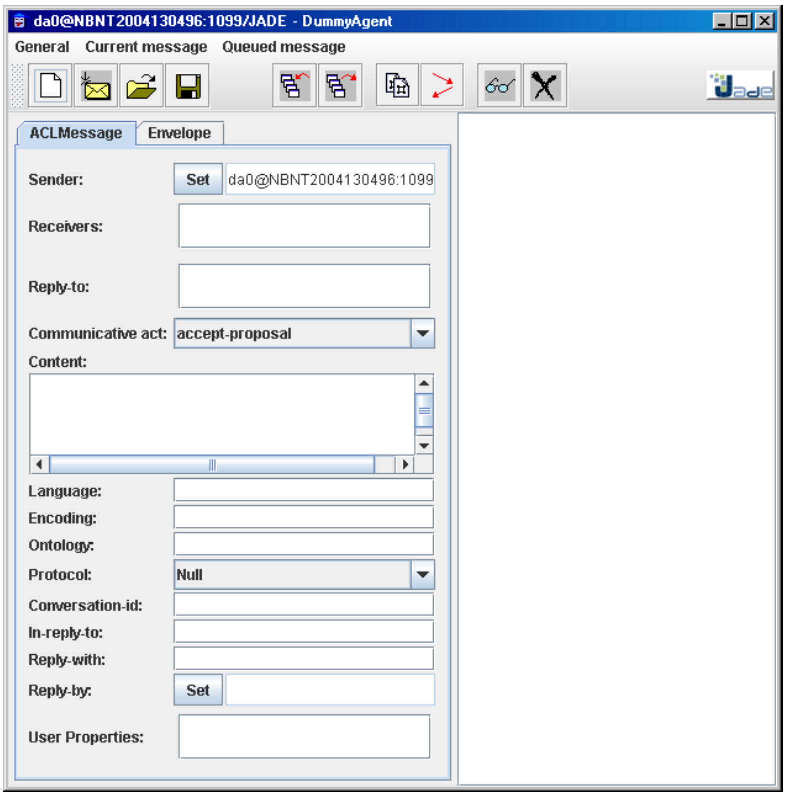


AgentPlatforms
ThisPlatform
Main-Container
ams@adambook:1099/JADE
UserAgent@adambook:1099/JADE
RMA@adambook:1099/JADE
df@adambook:1099/JADE
ServiceAgent@adambook:1099/JADE
sniffer0@adambook:1099/JADE
sniffer0-on-Main-Container@adambook:1099/JADE

REQUEST:301 ( )
REQUEST:2 (221 221)
INFORM:2 (221 253 221)
REQUEST:302 ( )
INFORM:10 (003)

17. Dummy agent.

It is a tool for monitoring and debugging. Creates a graphical interface. Using the GUI, we can create an ACL messages and send them to other agents. You can view all the messages sent and received.



Sender:	da0@NBNT2004130496:1099
Receivers:	
Reply-to:	
Communicative act:	accept-proposal
Content:	
Language:	
Encoding:	
Ontology:	
Protocol:	Null
Conversation-id:	
In-reply-to:	
Reply-with:	
Reply-by:	
User Properties:	

18. CompositeBehaviour

This class is composed of other behaviors (children). Operations are therefore defined in the "children" while the class itself deals with the scheduling of these operations according to specific rules. Class itself does not define these principles and gives only the interface. The rules must be defined in subclasses (SequentialBehaviour, ParallelBehaviour, FSMBehaviour).

19. Types of performative messages.

INFORM, REQUEST, AGREE, CANCEL, CONFIRM, DISCONFIM, FAILURE, UNKNOW, NOT_UNDERSTOOD, SUBSCRIBE and so on