Fabiola Dąbroś

Inżynieria Obliczeniowa gr.1

Ćwiczenia 7 – Zachowania

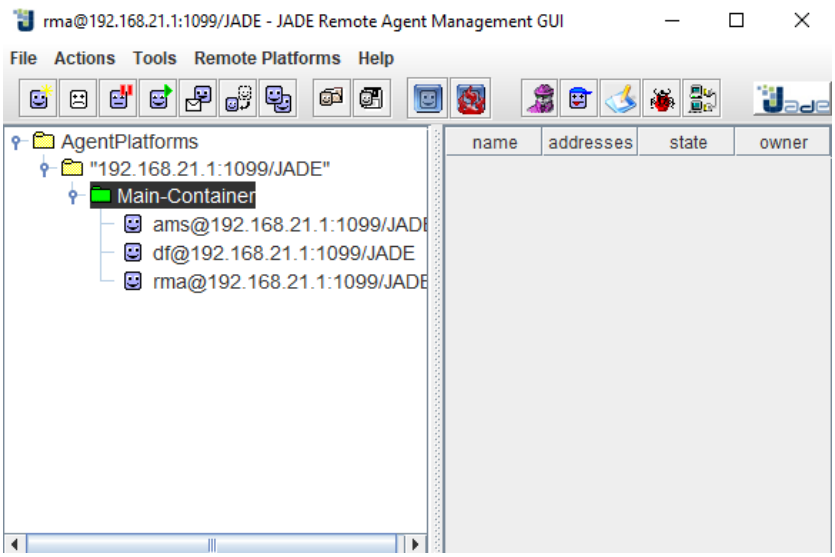
Wykonanie ćwiczenia rozpoczęłam od utworzenia klasy agenta o nazwie klasa_1. Agent ten powinien zawsze na samym początku wypisywać „startuje” , a następnie przed swoim usunięciem wypisywać „zaraz się usune”. Kod prezentuje się następująco:

```
import jade.core.Agent;
public class Klasa_1 extends Agent {

    protected void setup()
    {
        System.out.println("Startuje");


        System.out.println("Zaraz sie usune");
        doDelete();
    }

}
```

W wyniku czego otrzymujemy:

```
------------------------------------------
Startuje
Zaraz sie usune
```

Po czym widzimy, że agent nie jest dostępny w kontenerze.



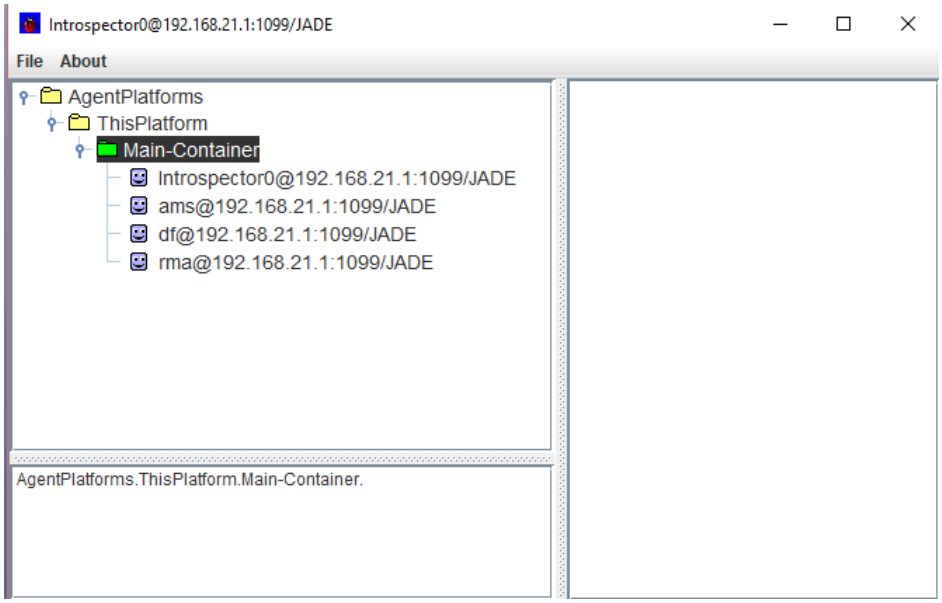
Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_2 na podstawie kodu Klasa_1. Do agenta należało dodać zachowanie polegające na jednokrotnym wykonaniu operacji wypisania na ekranie słowa „wykonuje”. W tym celu skorzystamy z klasy OneShootBehaviour, gdzie metoda action wykonuje się tylko raz. Kod prezentuje się następująco:

```
public class Klasa_2 extends Agent{

    protected void setup()
    {
        System.out.println("Startuje");

        addBehaviour(new OneShotBehaviour(this) {
              public void action() {
                  System.out.println( "wykonuje");
                System.out.println("Zaraz sie usune");
                doDelete();
              }
            });
    }

}
```

W wyniku czego otrzymujemy:

```
Startuje
wykonuje
Zaraz sie usune
```

Po uruchomieniu agenta introspektora nie zauważamy żadnych zachowań, ponieważ operacje są zbyt krótkie i szybkie.



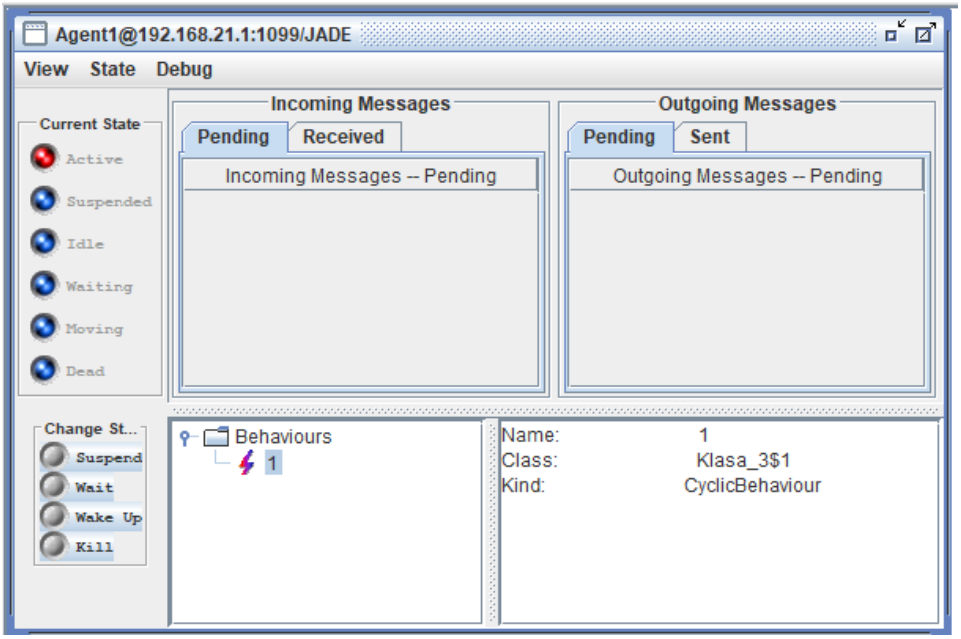
Kolejnym zadaniem było utworzenie klasy agenta o nazwie Klasa_3 na podstawie kodu Klasa_1. Do agenta należało dodać zachowanie polegające na wielokrotnym wykonaniu operacji wypisania na ekranie słowa „wykonuje”. W tym celu skorzystamy z klasy CyclicBehaviour, gdzie z każdym uruchomieniem wykonywane są te same operacje. Kod prezentuje się następująco:

```
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
public class Klasa_3 extends Agent{
    protected void setup()
    {
        System.out.println("Startuje");

    addBehaviour(new CyclicBehaviour(this) {
          public void action() {
            System.out.println("wykonuje");
          }
        });
}}
```

W wyniku czego otrzymujemy:

```
wykonuje
wykonuje
wykonuje
wykonuje
wykonuje
wykonuje
wykonuje
wykonuje
wykonuje
```

Tym razem po uruchomieniu agenta introspektora zauważamy, że agent wykonuje operacje w nieskończonej pętli.



Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_4 na podstawie kody Klasa_1. Do agenta należało dodać zachowanie generyczne, w którym istnieje możliwość wykonania różnych operacji w zależności od spełnionego warunku. W naszym przypadku polega ono na wykonaniu trzech kroków. W pierwszym kroku oraz w drugim kroku wypisuje konkretny komunikat , natomiast w kroku trzecim należy dodatkowo usunąć zachowanie z puli zachowań agenta. Kod prezentuje się następująco:

```
import jade.core.Agent;
public class Klasa_4 extends Agent {

    protected void setup()
    {
        System.out.println("Startuje");
        addBehaviour(new ThreeStepBehaviour());
    }
}

class ThreeStepBehaviour extends Behaviour{
     private int step = 0;
     public void action() {
         switch (step) {
            case 0:
                System.out.println("Krok pierwszy");
                step++;
                break;
            case 1:
                System.out.println("Krok drugi");
                step++;
                break;
            case 2:
                System.out.println("Krok trzeci");
                step++;
                break;
        }
     }

     public boolean done() {
         return step == 3;
     }
}
```

W wyniku czego otrzymujemy:

```
------------------------------------------
Startuje
Krok pierwszy
Krok drugi
Krok trzeci
```

Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_5 na podstawie kodu Klasa_1. Do agenta należało dodać zachowanie, które polega na pobieraniu z klawiatury liczby całkowitej. Jeśli użytkownik poda liczbę ujemną to zachowanie zostanie usunięte. Kod prezentuje się następująco:

```
import java.util.Scanner;
public class Klasa_5 extends Agent {

    protected void setup(){
        System.out.println("Startuje");
        addBehaviour(new NumberBehaviour());
    }
}

class NumberBehaviour extends Behaviour{
    boolean finished = false;
    public void action()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Podaj liczbe: ");
        int number = scanner.nextInt();
        if(number < 0) {
            System.out.println("Liczba ujemna.  Zachowanie agenta zostaje usunięte. ");
            finished = true;
        }
        else if(number == 0) {
            System.out.println("Zero.  Zachowanie agenta nie zostaje usunięte. ");
        }
        else {
            System.out.println("Liczba dodatnia.  Zachowanie agenta nie zastaje usuniete. ")
        }
    }
    public boolean done()
    {
        return finished;
    }
}
```

W wyniku czego otrzymujemy:

```
-------------------------------------------
Startuje
Podaj liczbe:
5
Liczba dodatnia.  Zachowanie agenta nie zastaje usuniete.
Podaj liczbe:
0
Zero.  Zachowanie agenta nie zostaje usunięte.
Podaj liczbe:
-5
Liczba ujemna.  Zachowanie agenta zostaje usunięte.
```

Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_6 na podstawie kodu Klasa_5. Kod należało zmodyfikować w taki sposób, aby zachowanie zawsze na początku wypisywało „zachowanie startuje” a na samym końcu „zachowanie zakończone”. W tym celu skorzystamy z metod onStart() oraz onEnd(), które wykonają się odpowiednio na początku, gdy agent doda już swoje zachowanie oraz na końcu gdy wykona się metoda done.
Kod prezentuje się następująco:

```
public class Klasa_6 extends Agent {

    protected void setup(){
        System.out.println("Startuje");

        NumberBehaviour nb = new NumberBehaviour() {
            public int onEnd() {
                System.out.println("zachowanie zakończone");
                //myAgent.doDelete();
                return super.onEnd();
            }

            public void onStart() {
                System.out.println("zachowanie startuje");

            }
        };

        addBehaviour(nb);
    }
}
```

W wyniku czego otrzymujemy:

```
-----------------------------------------
Startuje
zachowanie startuje
Podaj liczbe:

5
Liczba dodatnia.  Zachowanie agenta nie zastaje usuniete.
Podaj liczbe:
0
Zero.  Zachowanie agenta nie zostaje usunięte.
Podaj liczbe:
-2
Liczba ujemna.  Zachowanie agenta zostaje usunięte.
zachowanie zakończone
```

Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_7 na podstawie kodu Klasa_4. Do istniejącego zachowania generycznego należało dodać dwa kolejne.

Pierwsze jest wykonywane na poziomie metody setup() i wypisuje „pierwsze”.
Drugie jest dodane z poziomu zachowania generycznego. Jest dodane w pierwszym kroku i ma wypisac „drugie” .

Kod prezentuje się następująco:

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;

public class Klasa_7 extends Agent{

    protected void setup()
    {
        System.out.println("Agent startuje");
        addBehaviour(new Behaviour()
                        {
                            boolean finished = false;
                            public void action()
                            {
                                System.out.println("pierwsze");
                                finished = true;
                            }
                            public boolean done()
                            {
                                return finished;
                            }
                        });
        addBehaviour(new NewTickBehaviour(this));
    }

}

class NewTickBehaviour extends Behaviour{
     private int step = 0;
     Agent agent;
     public NewTickBehaviour(Agent a)
     {
         super(a);
         agent = a;
     }
```

```
public void action() {
    switch (step) {
       case 0:
           agent.addBehaviour(new Behaviour()
                   {
                       boolean finished = false;
                       public void action()
                       {
                           System.out.println("drugie");
                           finished = true;
                       }
                       public boolean done()
                       {
                           return finished;
                       }
                   });
           System.out.println("Krok pierwszy");
           step++;
           break;
       case 1:
           System.out.println("Krok drugi");
           step++;
           break;
       case 2:
           System.out.println("Krok trzeci");
           step++;
           break;
   }
}
```

W wyniku czego otrzymujemy:

```
------------------------------------------

pierwsze
Krok pierwszy
Krok drugi
drugie
Krok trzeci
```

Na samym początku otrzymujemy to co było w metodzie setup() czyli „pierwsze”. Kolejno wykonuje się „krok pierwszy” z drzewa zachowań, następnie „Krok drugi” i dopiero po nim następuje wypisanie „drugi”, które znajdowało się krok wyżej.

Kolejnym krokiem było utworzenie klasy agenta o nazwie Klasa_8 na podstawie kodu Klasa_1. Do agenta należało dodać zachowania, które spowodują kolejno wypisanie „maly tick” co dwie sekundy, „Duzy tick” co pięć sekund. Oprócz tego po 50 sekundach zostanie usunięte zachowanie odpowiedzialne za Duzy tick a po 100 sekundach zostanie usunięty cały agent.

W tym celu skorzystamy z zachowania TickerBehaviour oraz utworzymy własne zachowanie. Oprócz tego skorzystamy z metody removeBehaviour.

Kod prezentuje się następująco:

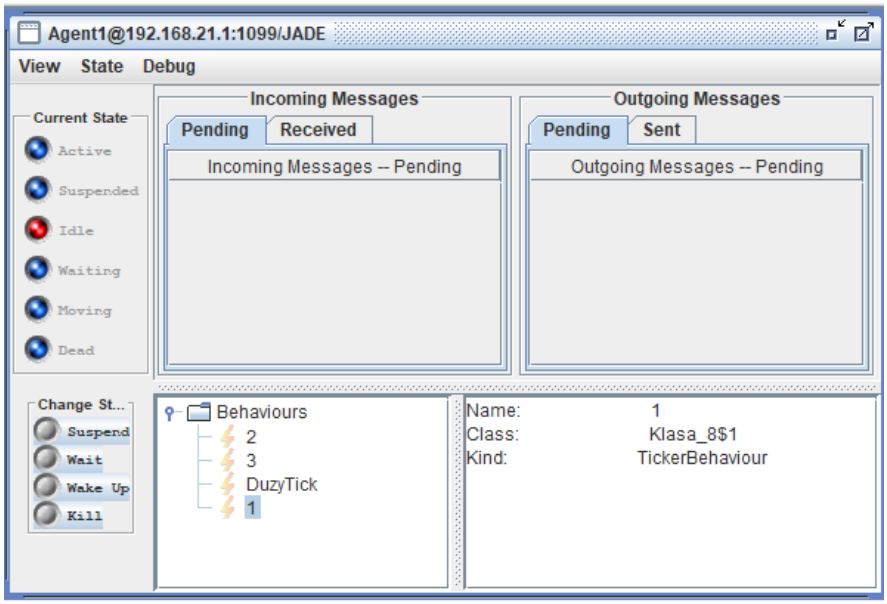
```
import jade.core.Agent;
public class Klasa_8 extends Agent {

    protected void setup()
    {
        System.out.println("Startuje");

        // tick co 2 sekundy
        addBehaviour(new TickerBehaviour(this, 2000) {
              protected void onTick() {
                System.out.println("maly tick");
              }
            });

        //tick co 5 sekund
        DuzyTick duzy = new DuzyTick(this,5000);
        addBehaviour(duzy);

        //usuniecie zachowania DuzyTick
        addBehaviour(new TickerBehaviour(this,50000)
                {
                    protected void onTick()
                    {
                        removeBehaviour(duzy);
                    }
                });


        //usuniecie calego agenta
        addBehaviour(new WakerBehaviour(this, 100000) {
              protected void handleElapsedTimeout() {
                System.out.println("Agent "+myAgent.getLocalName()+": It's wakeup-time. Bye.
                myAgent.doDelete();
              }
```

```
class DuzyTick extends TickerBehaviour
{
    public DuzyTick(Agent a, long period) {
        super(a, period);
    }

    protected void onTick()
    {
        System.out.println("Duzy tick");
    }
}
```

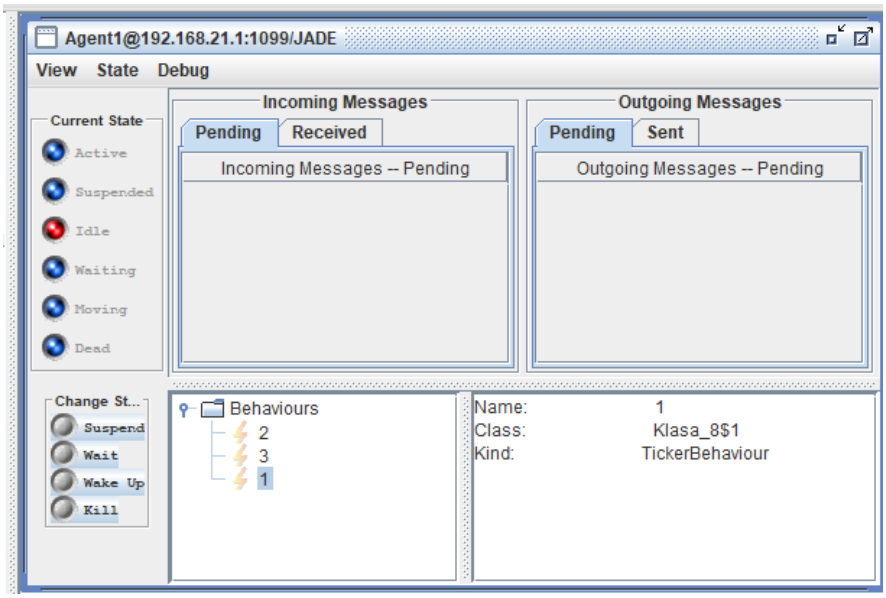
W wyniku czego otrzymujemy:

```
Duzy tick
maly tick
maly tick
maly tick
Duzy tick
maly tick
maly tick
maly tick
maly tick
maly tick
maly tick
maly tick
maly tick
maly tick
```

Podczas analizy introspektorem zauważamy, że początkowo mamy dostępny cztery zachowania.



Po upływie 50 sekund zostaje usunięte zachowanie Duży tick w wyniky czego pozostają nam trzy zachowania.



Następnie po 100 sekundach usuwa się cały agent w wyniku czego w introspektorze nie ma już nic.