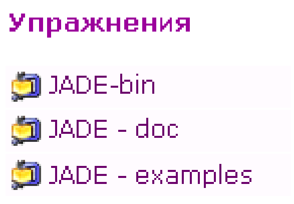
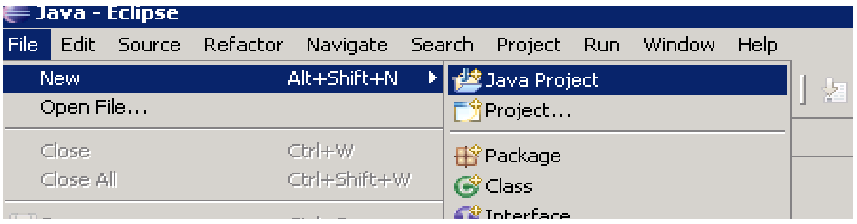
Упражнение 1
Агент-базирани технологии

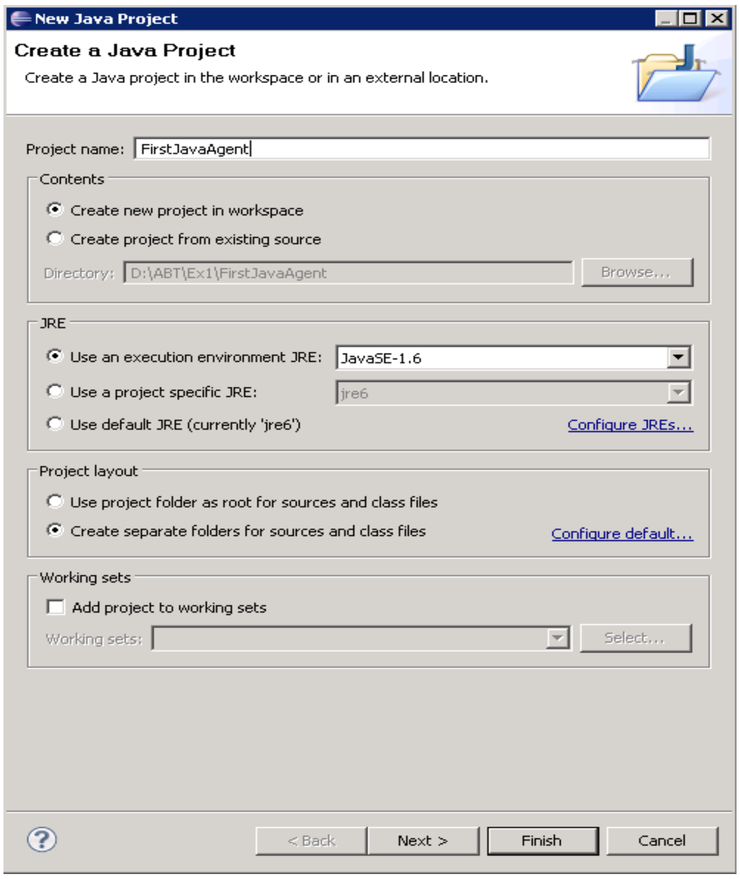
1. Ако нямате потребителско име, регистрирайте си в системата Moodle: http://www.csconf.org/ComputerSystems/
2. Ключовата дума за курса е:
   ABT2010
3. След като влезете в курса свалете следните три файла:



4. Разархивирайте ги в подходяща папка.
5. Стартирайте Eclipse (c:\Java\Eclipse)
6. Започнете нов проект



7. Задайте има на проекта


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: FirstJavaAgent

Contents
- (●) Create new project in workspace
- ( ) Create project from existing source

Directory: D:\ABT\Ex1\FirstJavaAgent [Browse...]

JRE
- (●) Use an execution environment JRE: JavaSE-1.6
- ( ) Use a project specific JRE: jre6
- ( ) Use default JRE (currently 'jre6') Configure JREs...

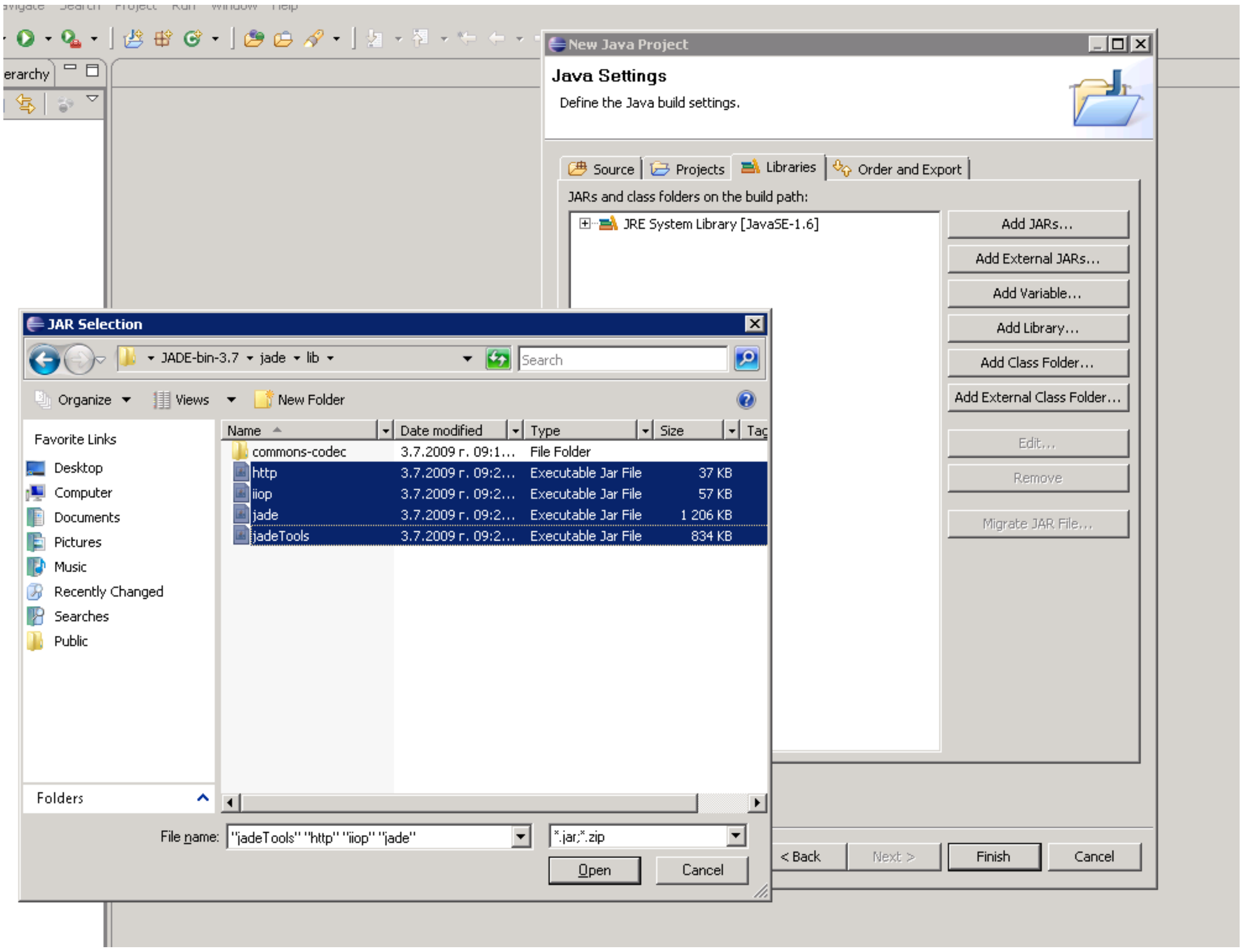
Project layout
- ( ) Use project folder as root for sources and class files
- (●) Create separate folders for sources and class files Configure default...

Working sets
- [ ] Add project to working sets

Working sets: [Select...]

< Back | Next > | Finish | Cancel

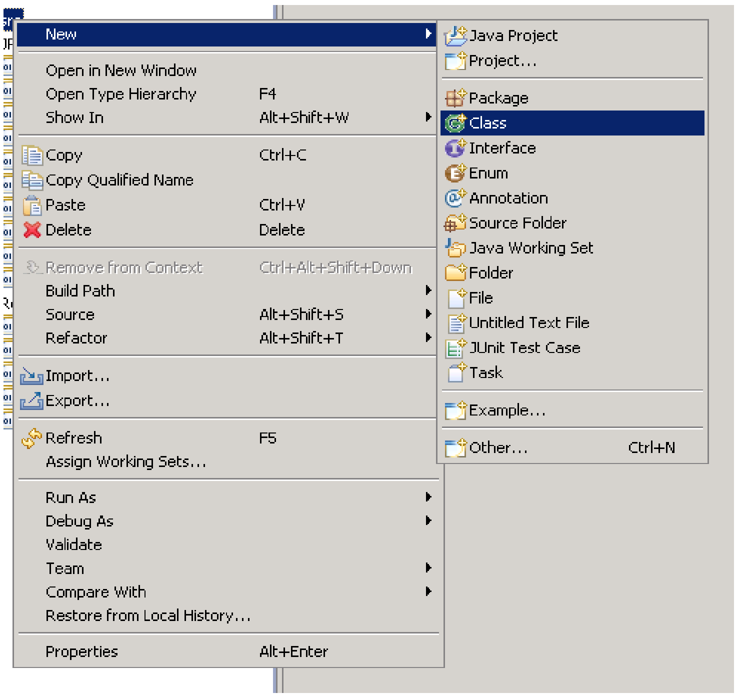
8. Добавете всички библиотеки (jar) от папката JADE-bin-3.7. Използвайте бутона „Add Extarnal JARs"



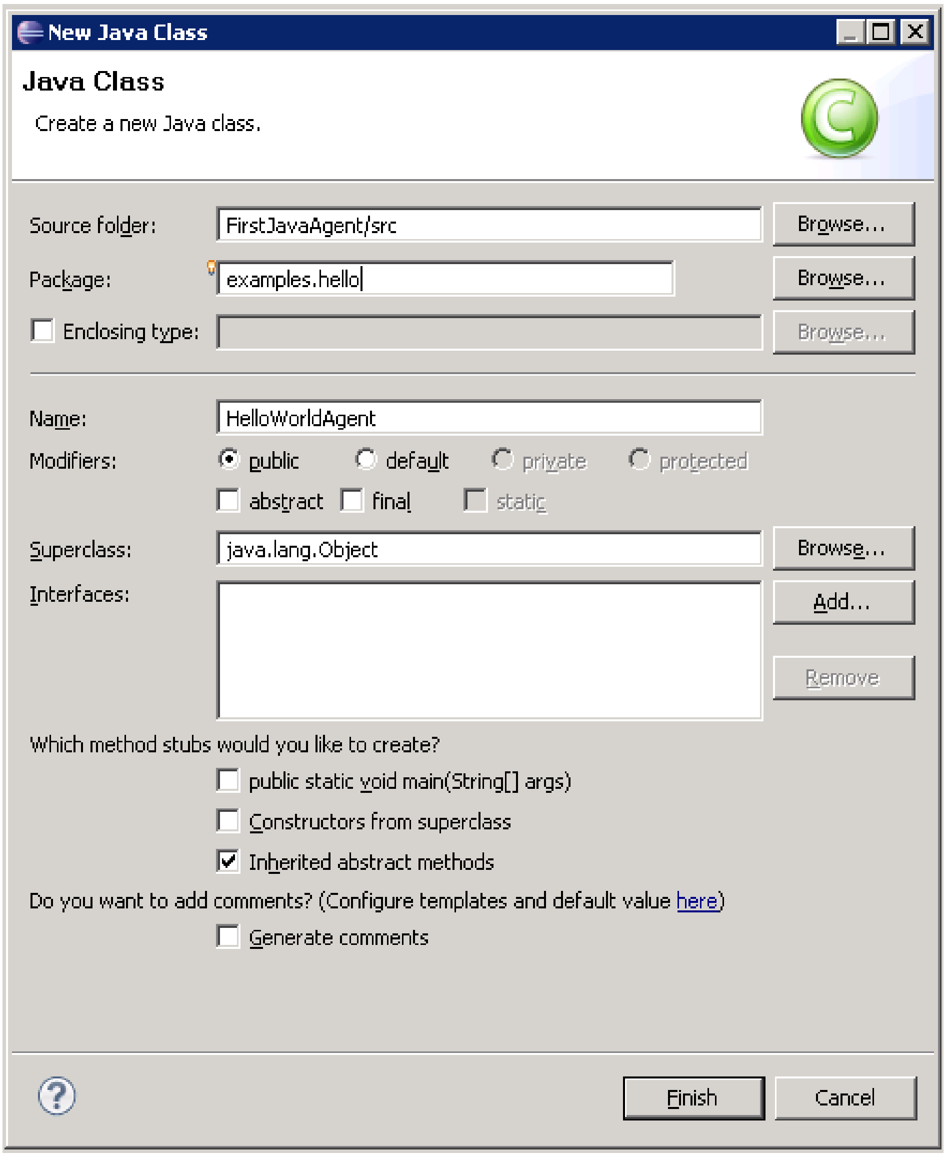
Name	Date modified	Type	Size
commons-codec	3.7.2009 г. 09:1...	File Folder	
http	3.7.2009 г. 09:2...	Executable Jar File	37 KB
iiop	3.7.2009 г. 09:2...	Executable Jar File	57 KB
jade	3.7.2009 г. 09:2...	Executable Jar File	1 206 KB
jadeTools	3.7.2009 г. 09:2...	Executable Jar File	834 KB

File name: "jadeTools" "http" "iiop" "jade"   *.jar;*.zip   Open   Cancel

9. Добавете нов клас:



10. Задайте име на пакета и класа:


New Java Class

Java Class

Create a new Java class.

Source folder: FirstJavaAgent/src

Package: examples.hello

[ ] Enclosing type:

Name: HelloWorldAgent

Modifiers: [● public] ○ default ○ private ○ protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

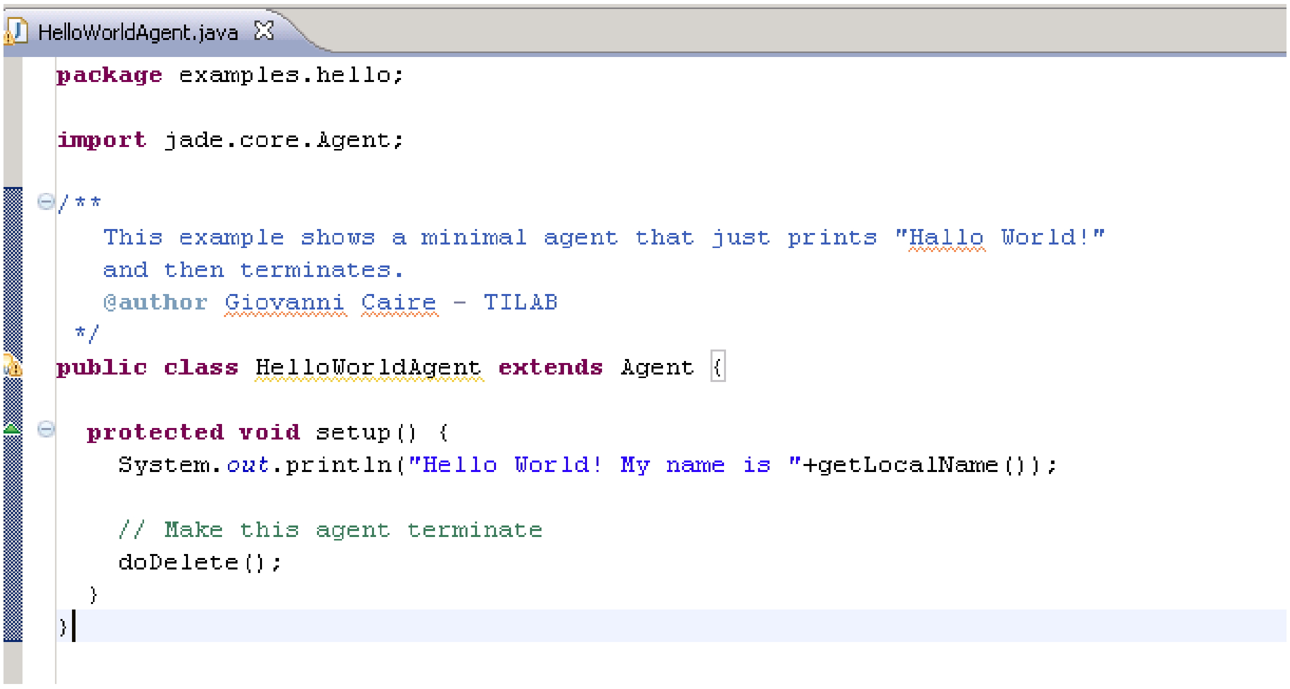
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

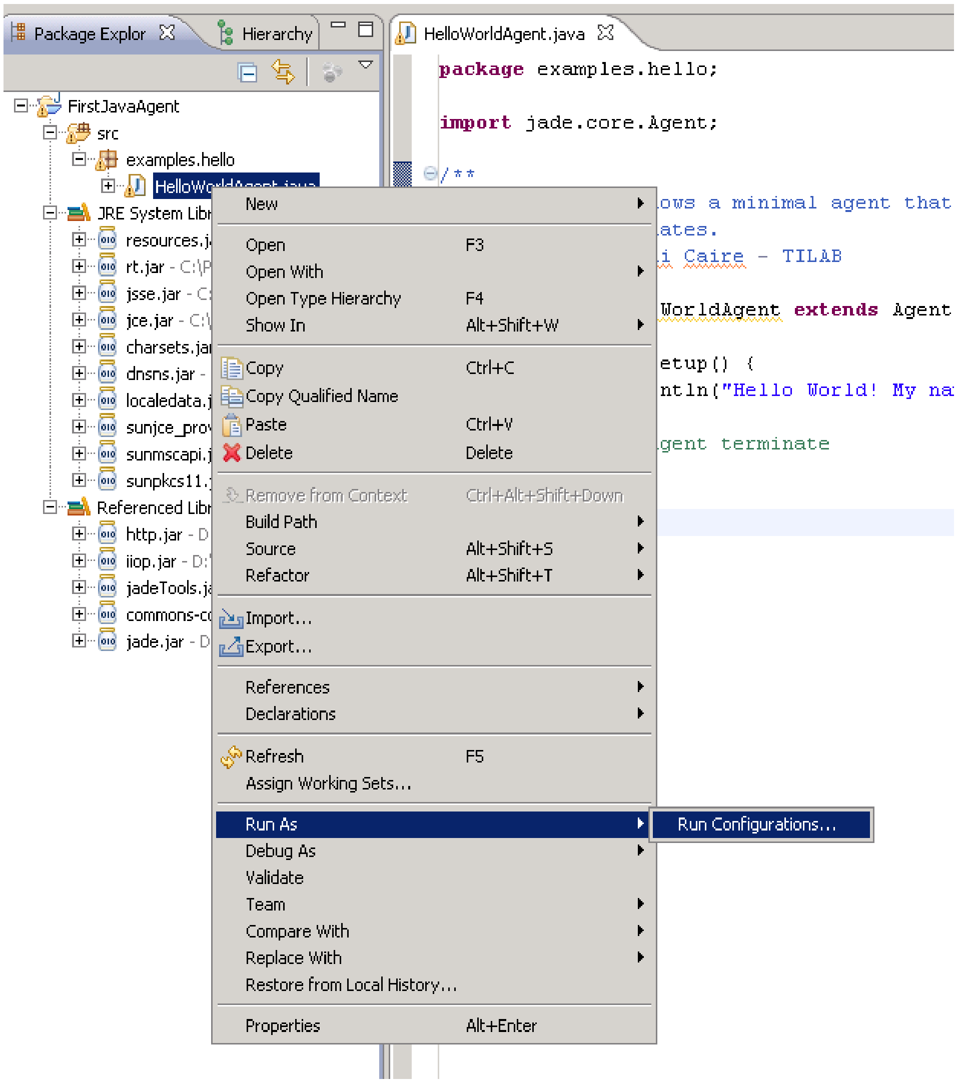
[ ] Generate comments

11. Въведете следния код:

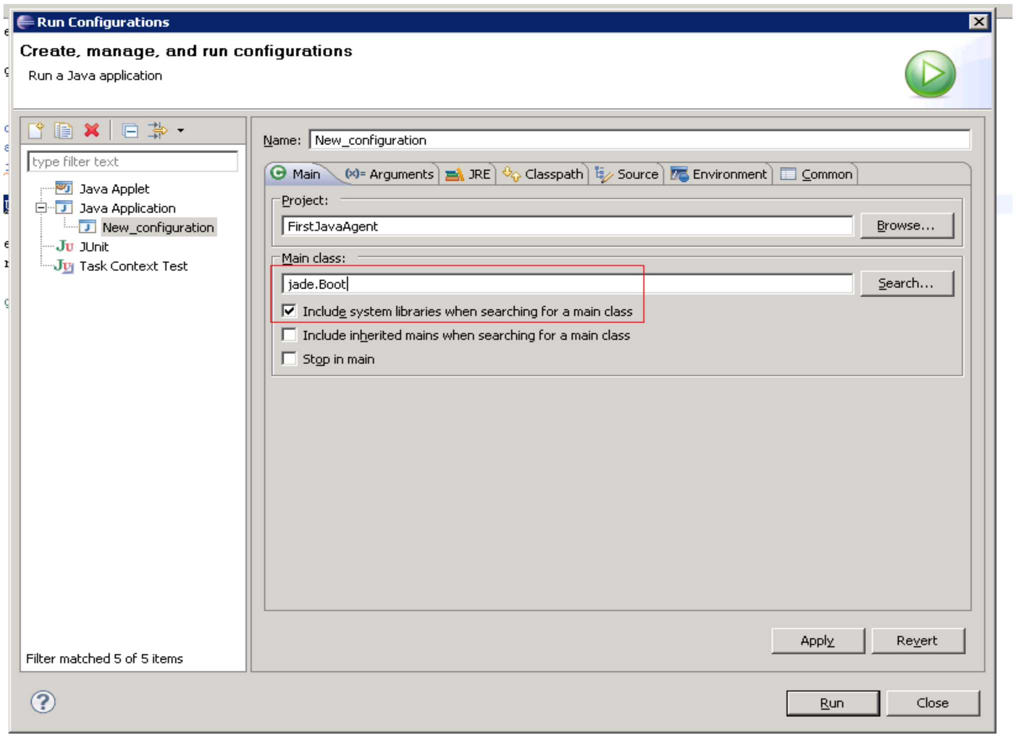


```
package examples.hello;

import jade.core.Agent;

/**
   This example shows a minimal agent that just prints "Hallo World!"
   and then terminates.
   @author Giovanni Caire - TILAB
 */
public class HelloWorldAgent extends Agent {

  protected void setup() {
    System.out.println("Hello World! My name is "+getLocalName());

    // Make this agent terminate
    doDelete();
  }
}
```

12. Стартиране на агента (Run -> Run Configurations):



13. В опцията „main class“ е jade.Boot и сложете отметка на „Include libraries when searching for a main class“.



Name: New_configuration

Project: FirstJavaAgent

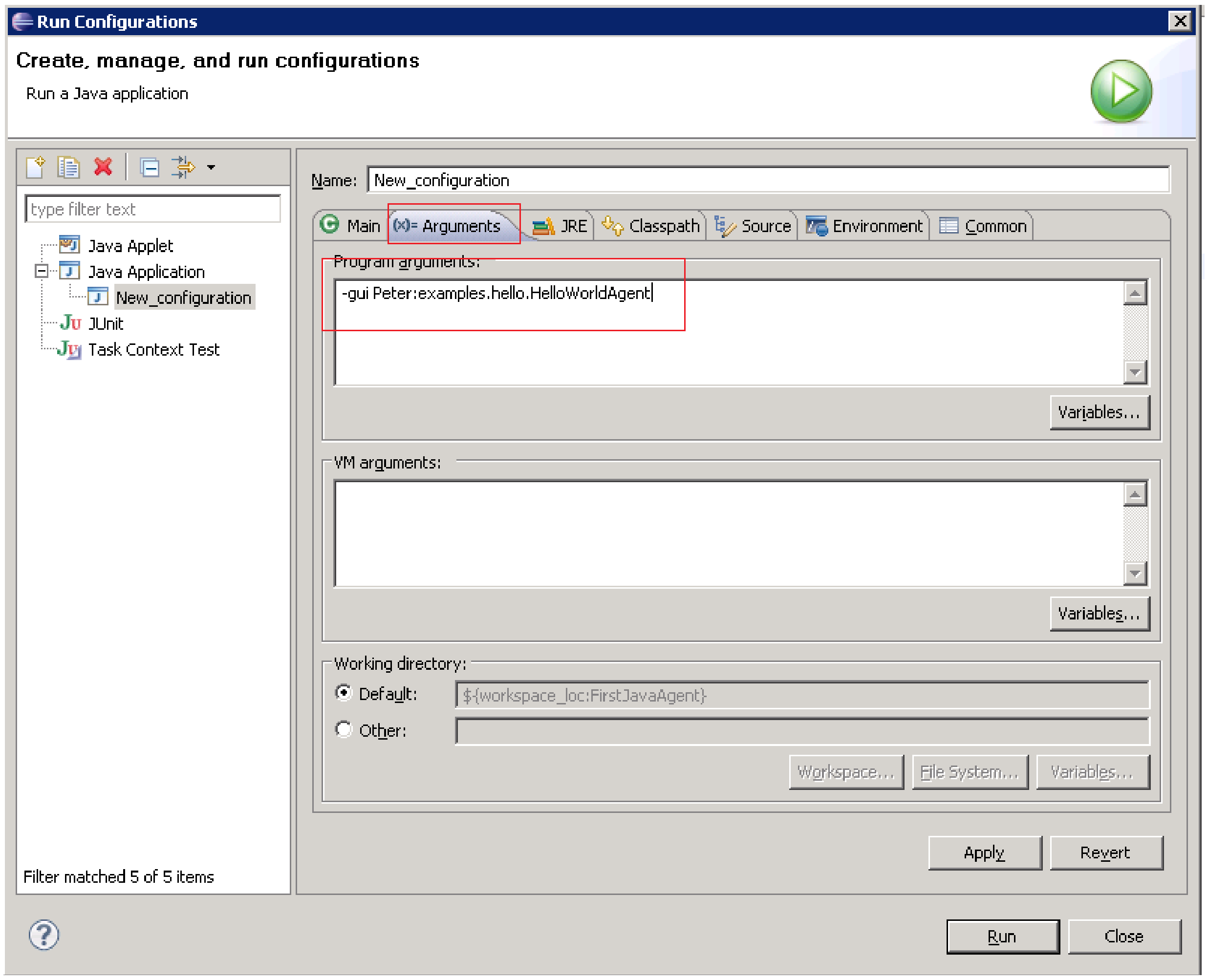
Main class: jade.Boot

[✓] Include system libraries when searching for a main class

[ ] Include inherited mains when searching for a main class

[ ] Stop in main

14. В страницата "argument" се записва командата:
    -gui jade.Boot [agent_nickname1:java_package1.agent_class1 agent_nickname2:java_package2.agent_class2 ...]



Name: New_configuration

Program arguments:

```
-gui Peter:examples.hello.HelloWorldAgent
```

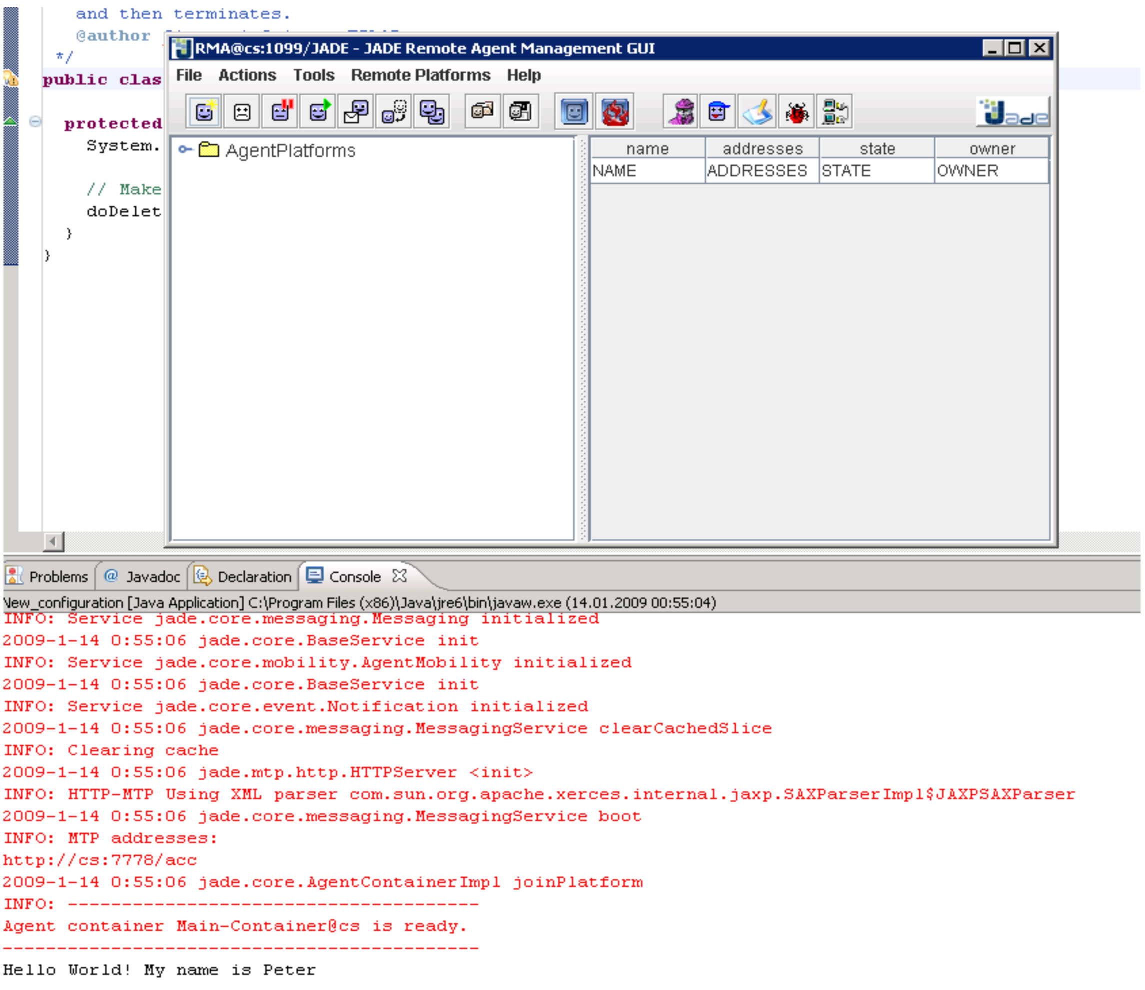
VM arguments:

Working directory:

(●) Default: ${workspace_loc:FirstJavaAgent}

( ) Other:

15. Резултата:



```
   and then terminates.
   @author
 */
public clas

  protected
    System.

    // Make
    doDelet
  }
}
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

```
New_configuration [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (14.01.2009 00:55:04)
INFO: Service jade.core.messaging.Messaging initialized
2009-1-14 0:55:06 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
2009-1-14 0:55:06 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
2009-1-14 0:55:06 jade.core.messaging.MessagingService clearCachedSlice
INFO: Clearing cache
2009-1-14 0:55:06 jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
2009-1-14 0:55:06 jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://cs:7778/acc
2009-1-14 0:55:06 jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Main-Container@cs is ready.
--------------------------------------------
Hello World! My name is Peter
```

Notes on this image

- JADE agent *platforms* have *containers* to hold agents. A platform can have many containers, not necessarily on the same computer. One container on a platform is "privileged". This *main container* resides on the host which also runs the platform's RMI server. Agents on various containers on a platform use the RMI protocol to communicate.
- The image above shows the GUI of the *Remote Monitoring Agent* (**RMA**) which appears when you use the -gui switch. In addition to itself, the RMA shows the presence of two other agents in the Main Container. The **ams** is the Agent Management System. An agent itself, it provides an environment with many services for agents on the platform. The **df** is the *Directory Facilitator*. It is an agent which provides a "yellow pages" for agents known to the platform.
- Agents residing on a platform must have unique names. A name is a "nickname" and an address separated by the at (@) sign. For example, RMA@IBM:1099/JADE is an agent with nickname RMA at the address IBM:1099/JADE. ("IBM" is the name of my Win2000 machine on a LAN.
- The addresses are in RMI format in this case. RMI is used for intra platform communication.

(CORBA or HTTP are used for inter platform communication.) The address consists of a host name, in this case IBM, and a port on which the RMI naming service is active, in this case, 1099, the default port for RMI,1099. The name JADE distinguishes JADE RMI invocations from other possible RMI services. Note that in this case, the host name does not have a domain attached. If you wanted a full name you can use the -host switch: java jade.Boot -gui -host jupiter.scs.ryerson.ca, for example. There is also a -port switch if you don't like 1099.

Задача 1: Реализирайте агент, който извежда информация за агента в следния формат:

```
Agent Started: Hello World!
-----About Me:-----
My local name is:<име на агента>
My globally unique name is:<глобално име на агента>
-----About Here:-----
I am running in a location called:<име на контейнера, в който е стартиран>
Which is identified uniquely as:<уникалния индентификатор>
And is contactable at:<адреса>
Using the protocol:<протокола>
```

***Помощ*:**
1. За да разпечатите информацията след надписа
-----About Me:-----
използвайте методите:
`getLocalName()`
`getName()`

2. За да разпечатите информацията след надписа
-----About Here:-----
използвайте класа Location и неговите методи

Abstract interface to represent JADE network locations. This interface can be used to access information about the various places where a JADE mobile agent can migrate.

Method Summary	
java.lang.String	**getAddress**() Read the address for a location.
java.lang.String	**getID**() Read a unique ID for the location.
java.lang.String	**getName**() Read the name of a location.
java.lang.String	**getProtocol**() Read the protocol for a location.

Making your agent do stuff: Agent Behaviours.

The previous example doesn't actually do anything, all code is just run inside the *setup()* method. Typical agents implement complex behaviours, which may involve multiple simultaneous related or unrelated tasks, rather than forcing you to write multithreaded agents (with all of the problems that this can cause) jade introduces a system of *Behaviours* to assist you in building and re-using agent functionality.

Essentially when using Jade's behaviours each agent can be seen as a set of cooperatively multithreading processes ,that is to say that each process gets run, and performs a portion of its task before actively yeilding controll back to the process scheduler. While this may seem old-fashioned it adds a degree of determinism to writing agents, and allows for a fairly natural decomposition of the agents overall behaviour.

The skeleton of an typical agent class is shown below:

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

    public class myAgent extends Agent
    {
        protected void setup()
        {
            addBehaviour( new myBehaviour( this ) );
        }


        class myBehaviour extends SimpleBehaviour
        {
            public myBehaviour(Agent a) {
                super(a);
            }

            public void action()
            {
               //...this is where the real programming goes !!
            }

            private boolean finished = false;

            public boolean done() {
                return finished;
            }

        } // ----------- End myBehaviour

    }//end class myAgent
```

Here, the Behaviour is defined to be a subclass of **SimpleBehaviour**, the most flexible of jade's predefined Behaviours. It is also positioned to be an *internal* class to the Agent. It could also have been implemented as an *anonymous* class or defined in a *separate* file. In all cases, it is usual to pass a reference to the owner agent (*this*) when creating a Behaviour.

A last point is the provision of a mechanism to terminate the behaviour. In Jade, as long as a behaviour is not "**done**", its action method will be called repeatedly after every **event** - such as receipt of a message or expiry of a timer delay. The example shows a common technique used to terminate the behaviour: a FINISHED flag which is tested in the "*done*" method and can be set in "*action*".

Simple0.java

Here is a more typical implementation of the HelloAgent.

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

```
public class Simple0 extends Agent
{
     protected void setup()
     {
          addBehaviour( new B1( this ) );
     }
}

class B1 extends SimpleBehaviour
{
     public B1(Agent a) {
           super(a);
     }

     public void action()
     {
        System.out.println( "Hello World! My name is " +
                myAgent.getLocalName() );
     }

     private boolean finished = false;
     public  boolean done() {  return finished;  }

} //End class B1
```

The example also shows two new things:

- **myAgent**: a local variable of all Behaviour objects which stores the agent reference passed as a parameter when the behaviour was created.
- The class B1 is specified outside the context of the Agent, but it can use **myAgent** to access its owner's attributes and methods.

```
Hello World! My name is fred
    Hello World! My name is fred
    Hello World! My name is fred
    Hello World! My name is fred
```

безкраен цикъл

The problem here is that we haven't explicitely indicated that the action was "done".

Задача 2: Реализирайте горния агент така, че да приключи работа след 5-тото извикване.