**Distributed Network Application Development, Spring 2015**

Bucăţea Maria Cristina and Cârlig Radu Victor

June 23, 2015

Professor: Costin Bădică

Year of Study: Third Year

Group: 10305

Section: Computers in English

1 Problem statement

Mobile Code: Distributed calculus with resources from volunteers

It is considered the situation where a research group needs a lot of calculation power and their supporters want to give them acces to their home resources.

Implement a software client that can be installed by anyone to offer their personal computer resources. People who make their computers available to the research group will be named ‚volunteers’.

Implement a server to which the clients can connect automatically. This server will distribute tasks to clients as code that is going to be run on the volunteer’s computer. After the code is run, the client will send to the server the result.

2 Pseudocode

**Server:**

StartListening()

1. **while** tasks are available **do**

2. switch clientRequest

3. case requestTasksList

4. SendTasksList

5. case chooseTask

6. SendTask

7. case fileOutput

8. print result

SendTask()

1. **if** tasks are available **do**

2. send fileInfo

3. send javaFile

4. remove task from available tasks list

SendTasksList()

1.GetTasksList

2. **if** tasks are available **do**

3. send tasksList

4. **else do**

5. send tasksUnavailable response

**Client:**

StartListening()

1. **while** tasks are available **do**

2. **if** not busy **do**

3. getTasksList

getTasksList()

2. **switch** serverResponse

3. **case** tasks available

4. receive list of tasks

5. sort tasks

6. choose task

7. getFile

8. **case** tasks unavailable

9. return

getFile()

1. receive fileInfo

2. receive java file

3. executeFile

executeFile(filename, extension, arguments)

1. compile file

2. run file

3. get result

4. sendResult

5. delete file

3 Application design

The application is divided into two main components: server and client. Each of these components are java files (.java). The project contains some additional java files which represent the tasks that the server can send to clients.

The clients are able to connect to the server and send requests. The supported requests are: request the list of available tasks, request a certain task, give the result of the task to server.

The server can respond to clients. The responses that are supported are: the requested task is available, the requested task is unavailable; the server is also able to send the list of available tasks and to send the file that represents the task requested if available.

**Server functionality:**

* void StartListening();

This method sets up the connection between server and clients. It receives the requests from clients and calls the appropriate functions to deal with the requests. The method also receives the result of the taks sent by a client and prints it in the console.

* void CreateTasks();

This is the method where the list of tasks is created and mapped. For each task available there is a java file that represents the code that will be sent to clients.

* String GetTasksList();

This method concatenates all the available tasks; on a line are the three characteristics of a task: id, estimated complexity and the date when it was added. Is is used when a client requests the list of tasks to obtain the concatenated string in order to be sent to the client.

* void SendTasksList();

This method is responsible if sending the list of available tasks to the client that requested it. It verifies if there are tasks available and sends the appropriate response to the client and if there are tasks available, it gets the concatenated string by calling the GetTasksList method and sends it through a DataOutputStream to the client.

* void SendTask();

This method is responsible for sending the java file for the corresponding task that was requested by a client. It first receives the id of the task requested and then it retrieves and sends to the client the information about that task (name of the file corresponding to the task, the extension of the file and the arguments needed to run the file). Afterwards, the file is sent to the client and it is removed from the list of available tasks.

* Task getTaskFromString (String task);

This method receives an available tasks as a concatenated string. It transforms the data into a Task object and returns the task.

* void main(String args[]);

The main function creates a server object and starts listening to requests from clients.

**Client functionality:**

* void startListening();

This method communicates with the server. When the client is not busy, it requests makes requests while there are still tasks available on the server. Requests are made by calling the getTasksList method that receives the list of tasks available, chooses a task, runs the code and returns the result. When there are no more tasks available, the method returns and it means that all the tasks were processed and the results were sent to the server.

* void restartStreams();

This method restarts the streams for the client-server communication.

* void getTasksList();

This method collects the list of available tasks from server. It also sorts the tasks and chooses the apropiate task to be requested from the server by calling the getFile method. The client is now set to ‚busy’ until the task is done.

* void getFile();

This method requests from server the chosen task to be resolved. Then, it receives the info about the task and the file to run. After it finishes receiving, it calls the executeFile method where the code is executed and the result is received.

* void executeFile(String filename, String extension, String arguments);

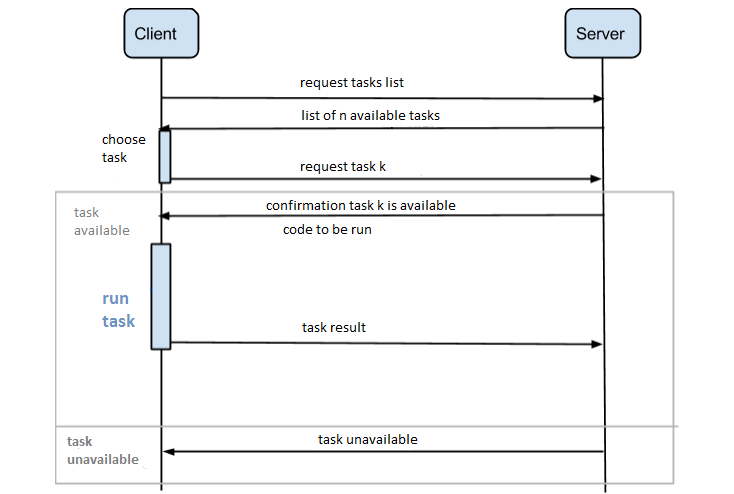
This method compiles and runs the file corresponding to the current task. After it receives the result, it calls the sendResult method which sends the result to the server. It also deletes the file received for the current task, since the result was received and the file is no longer needed.

* void sendResult(String result);

This method receives the result of the current task and sends it to the server from which it was requested.

* static void main(String args[]);

The main function creates a new client object and then it call the startListening method in order to communicate with the server.



In the above picture is explained the communication protocol between the server and a client for allocating a task.

The list of tasks consists of a number of n tasks represented by an id, complexity and the date when it was added. After receiving the list of tasks, the client sorts it and chooses the preferred task. The client then requests the chosen task from the server which responds by telling if the task is available or not.

If the requested task is available, the client receives the file containing the code to be run, so it compiles and runs the file in order to obtain the result. When the result is complete, it is send back to the server.

The client continues to request tasks from the server until there are no more tasks available (the server will send a response telling the client when the list of tasks is empty).

4 Experiments and results

Inputs:

Tasks.txt

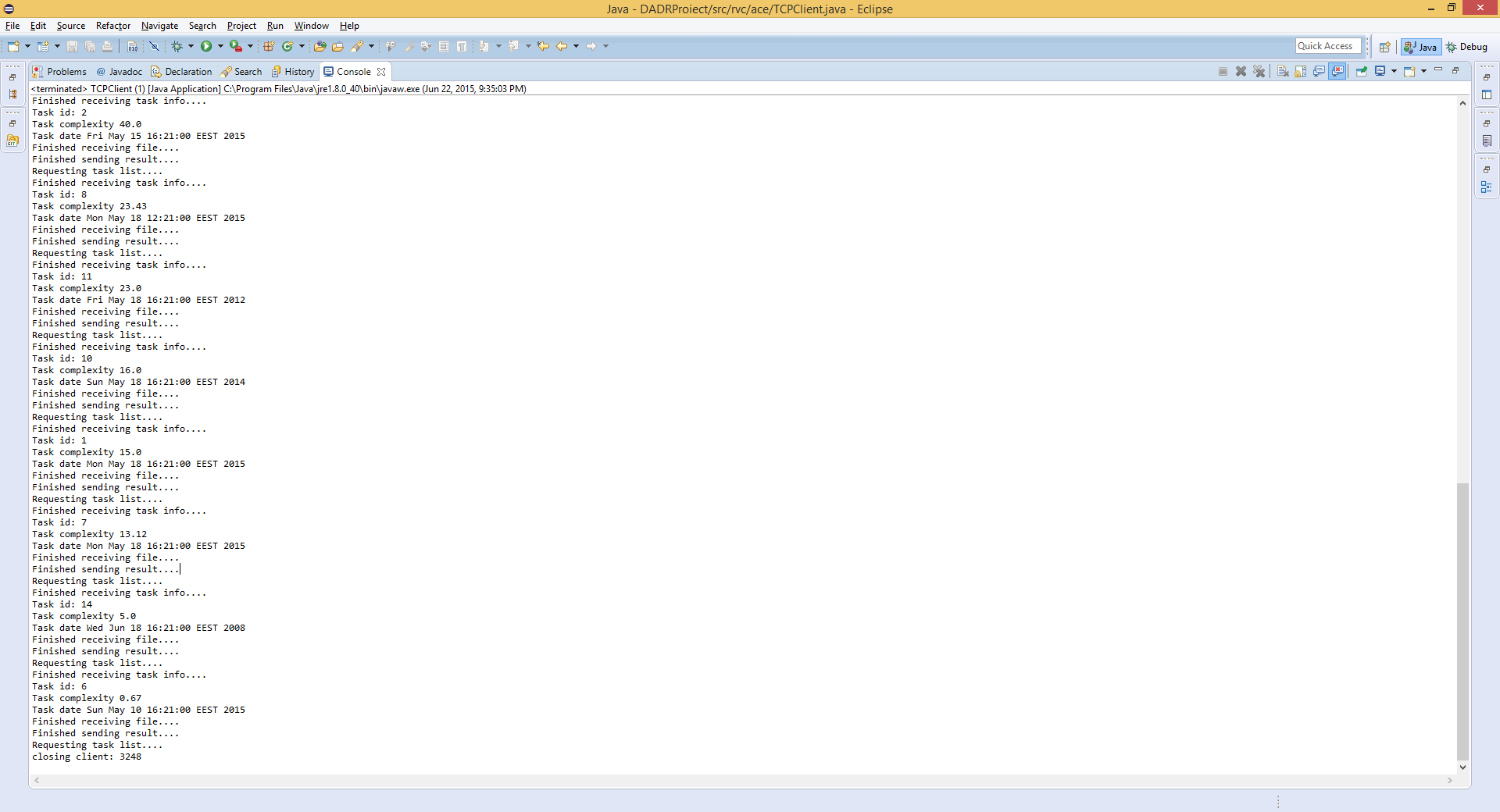
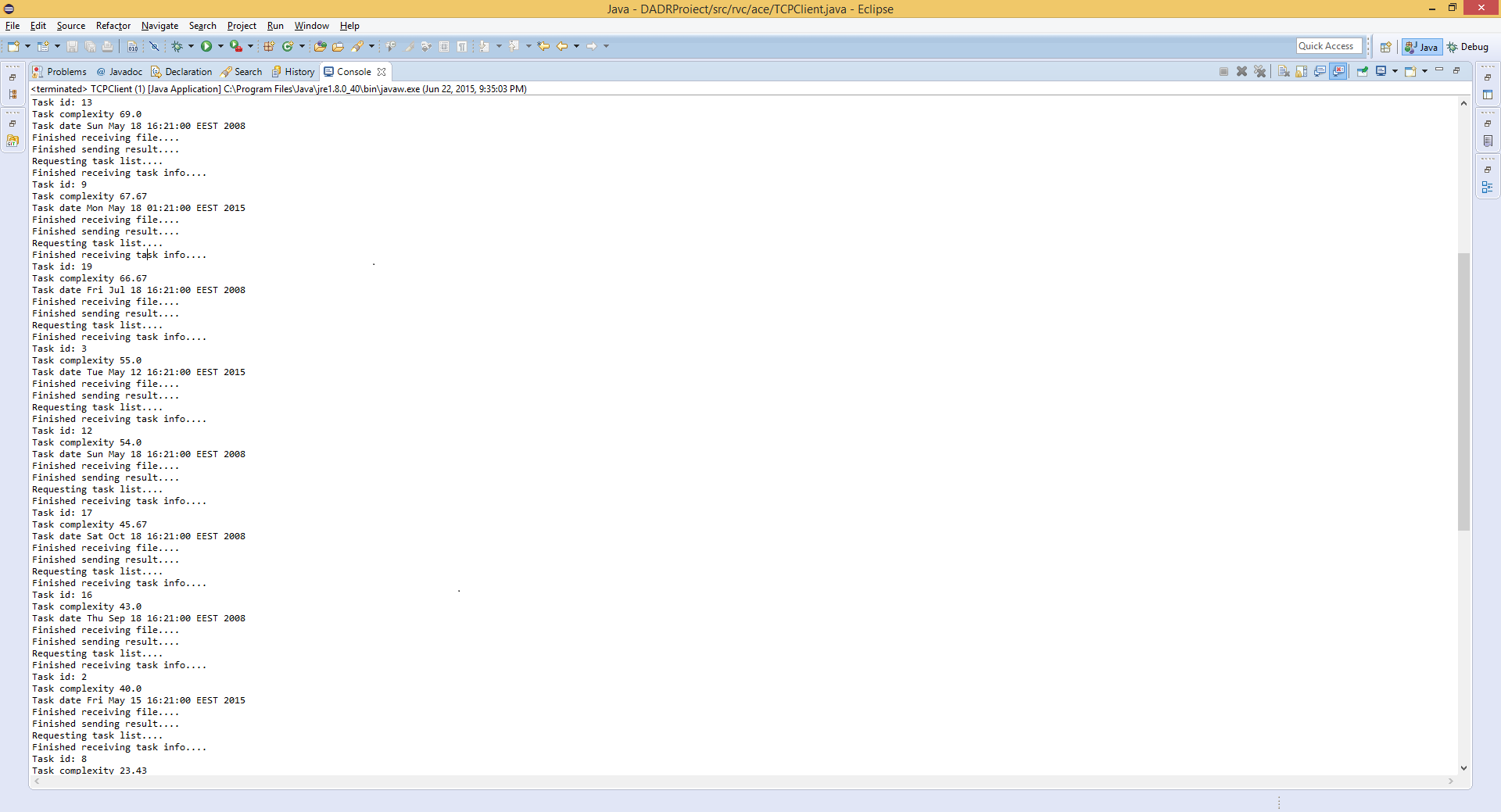
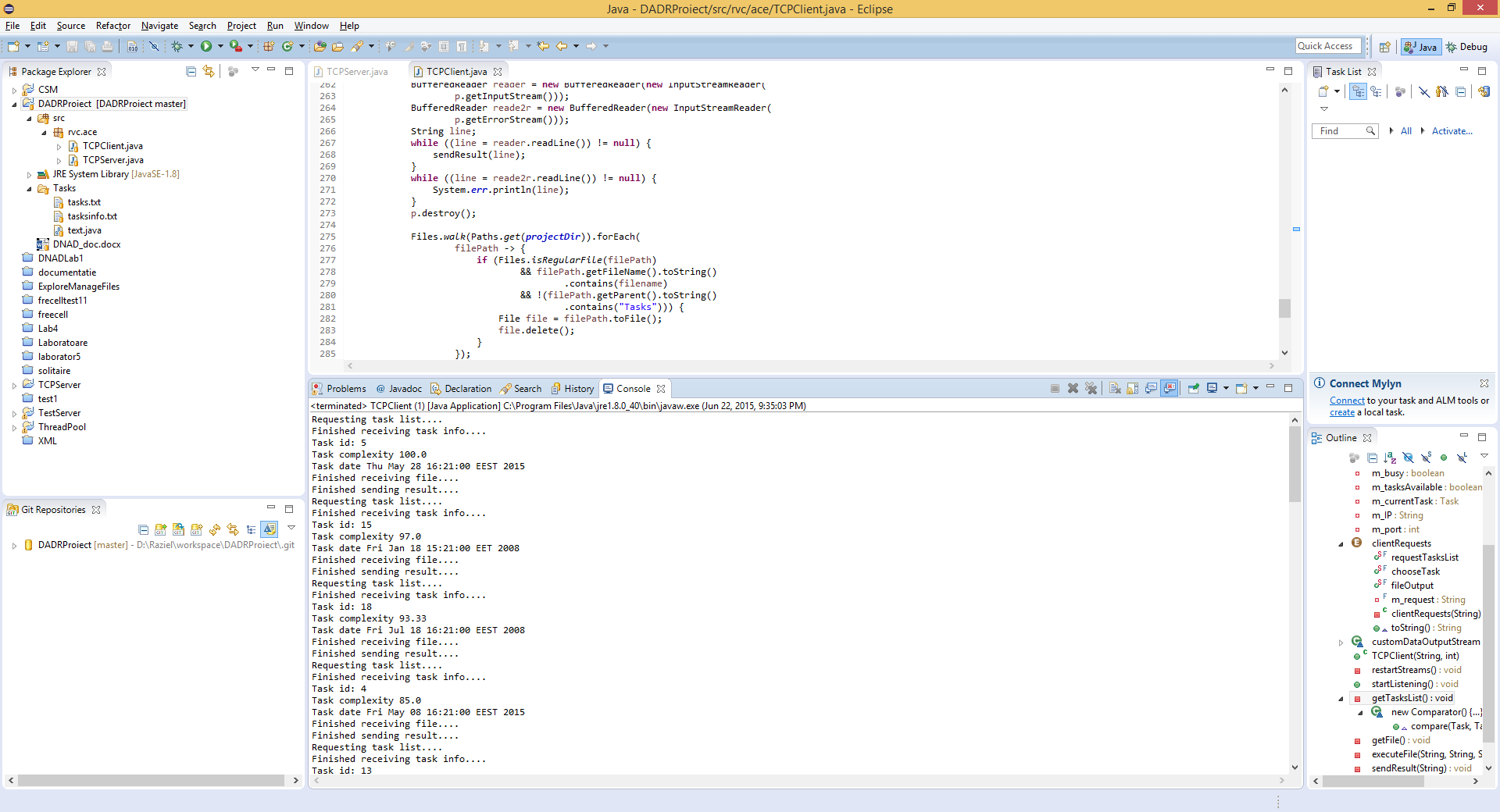


TasksInfo.txt

****

**Client results**

The client takes the most complex task from the list received from the server and executes them.



**Server results**

