|  |
| --- |
| Distributed systems |
| Remote Methods Invocation |
| Master M1 MOSIG, Grenoble Universities |
| GRÄBNER Laurent |
| TCHOUGOURIAN Tigran |
|  |
| **22/02/2011** |



|  |
| --- |
| **Supervisor :**  BOUCHENAK Sara |

|  |
| --- |
| The assignment is about building an RMI-based chat system, where participants can dynamically join, leave, exchange messages. |

Table of contents

[1. Implementing a thread-safe linked list 2](#_Toc278742704)

[1.1. Question II.1 2](#_Toc278742705)

[1.1.1. What result could you expect? 2](#_Toc278742706)

[1.1.2. Why it doesn’t fail? 2](#_Toc278742707)

[1.1.3. Validity of the test? 3](#_Toc278742708)

[1.2. Question II.2 3](#_Toc278742709)

[1.3. Question II.3 4](#_Toc278742710)

[1.4. Question II.4 4](#_Toc278742711)

[1.5. Question II.5 5](#_Toc278742712)

[2. Improving concurrency 5](#_Toc278742713)

[2.1. Question III.1 5](#_Toc278742714)

[2.2. Question III.2 6](#_Toc278742715)

[3. Improving fairness 7](#_Toc278742716)

[3.1. Question IV.1 7](#_Toc278742717)

[3.2. Question IV.2 7](#_Toc278742718)

[3.3. Question IV.3 7](#_Toc278742719)

[4. Annex 9](#_Toc278742720)

[4.1. Naïve implementation 9](#_Toc278742721)

[4.2. Improved concurrency implementation 9](#_Toc278742722)

[4.2.1. Reader\_writer structure 9](#_Toc278742723)

[4.2.2. Implementation 10](#_Toc278742724)

[4.3. Fairness implementation 11](#_Toc278742725)

[4.3.1. Reader\_writer structure 11](#_Toc278742726)

[4.3.2. Begin read 11](#_Toc278742727)

[4.3.3. End read 12](#_Toc278742728)

[4.3.4. Begin write 12](#_Toc278742729)

[4.3.5. End write 13](#_Toc278742730)

# Objects of the application

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Client | | - id : Integer  - server : Server  - nickname : String | | + register()  + unregister()  + send(msg : String)  + receive(msg : String) | |  | |  | | --- | | Server | | - clients : Client[] | | + add(c : Client)  + remove(c : Client)  + broadcast(msg : Message) | | |
|  | |  | | --- | | Message | | - text : String  - date : Date  - from : String | |  | | |  |

Every client is represented by an object client; the server is unique in our design and stores all the clients. The Message carries the information from one client to the others.

# Remote objects and regular Java object

|  |  |
| --- | --- |
| Require interface | Provided Interface |
| |  | | --- | | Client | | register()  unregister()  send(msg : String) | | |  | | --- | | Server | | add(c : Client)  remove(c : Client)  broadcast(msg : Message) | |
| |  | | --- | | Server | | broadcast(msg : Message) | | |  | | --- | | Client | | receive(msg : Message) | |

* Client and Server are remote objects.
* Client and Message has to be serializable.
* Message is a regular Java object, because no methods are called on that object remotely.

# Interfaces and classes of Remote objects

## Interfaces

## Classes

# Interfaces and classes of regular Java objects

## Interfaces

## Classes

# History