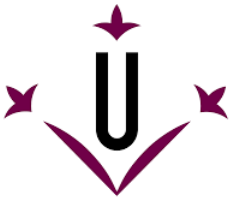


Universitat de Lleida

Activity 5

RMI Online Multi-Choice Exams

An essay by:
Guillem Felis de Dios & Sergi Sales Jové

Deadline:
10th of December 2020

Universitat de Lleida
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Distributed Computing

Teachers:
Josep Lluís Lérida
Eloi Gabaldón Ponsa

List of contents

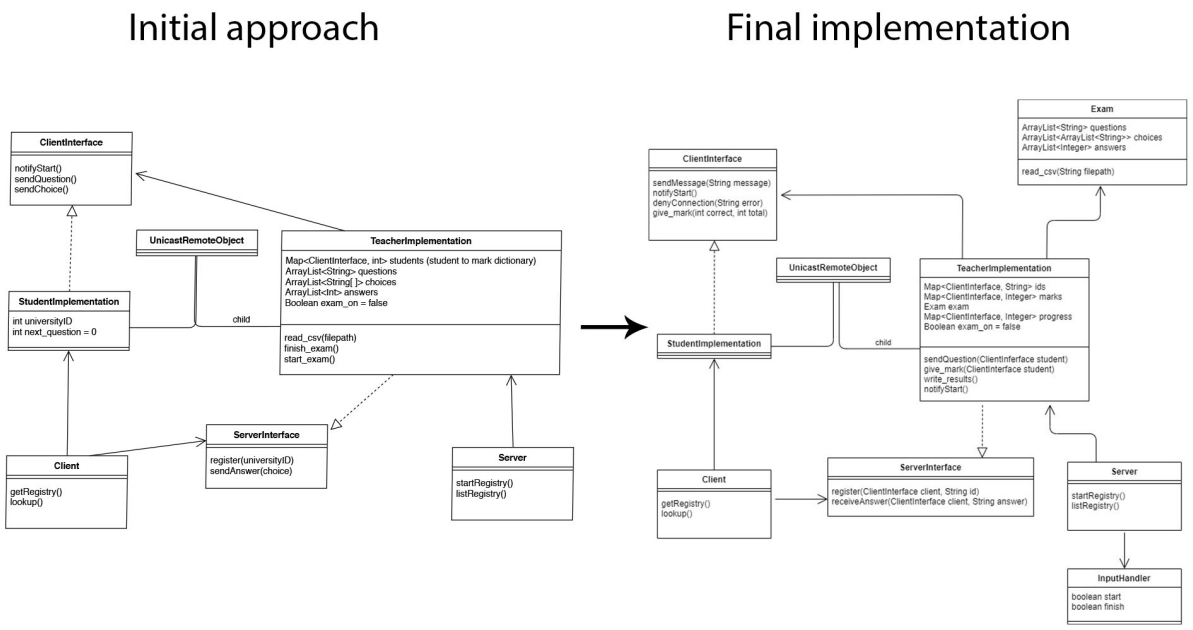
1 Abstract

We present a client-server application that performs multiple choice exams remotely, simulating the teacher as the server and the students connecting as the clients. We implemented this using Java RMI. In this document we will explain briefly the most important parts of this project and our take on it.
GitHub Repository: https://github.com/gfelis/udl-exam-RMI.git

2 Introduction

This year has been tainted by a major issue, COVID-19. To deal with it, lock-downs were established, and a lot of educational systems had to face the difficult choice of how to keep classes as normal as possible in on online environment. In doing so, we've learn a lot of new technologies that helped us learning and teaching, such as video-chat softwares. And this technologies rely on communication and connection. So, it was the perfect time to use this as an idea to implement our knowledge on Java RMI and its client-server structure.
Our assignment was to implement a client-server application to perform a multiple choice exams remotely. The idea was to develop this application simulating a situation where the teacher, working as a server, stays open until the beginning of the exam. Before that, the students, a.k.a. clients, must access the server and register using their university ID. Now, once the exam begins, no students can connect if they weren't before, and if someone disconnects during the exam, it cannot reconnect again.
While the exam is still in process, the teacher will be sending the questions and possible answers every time someone replies the previous one.

3 UML diagram



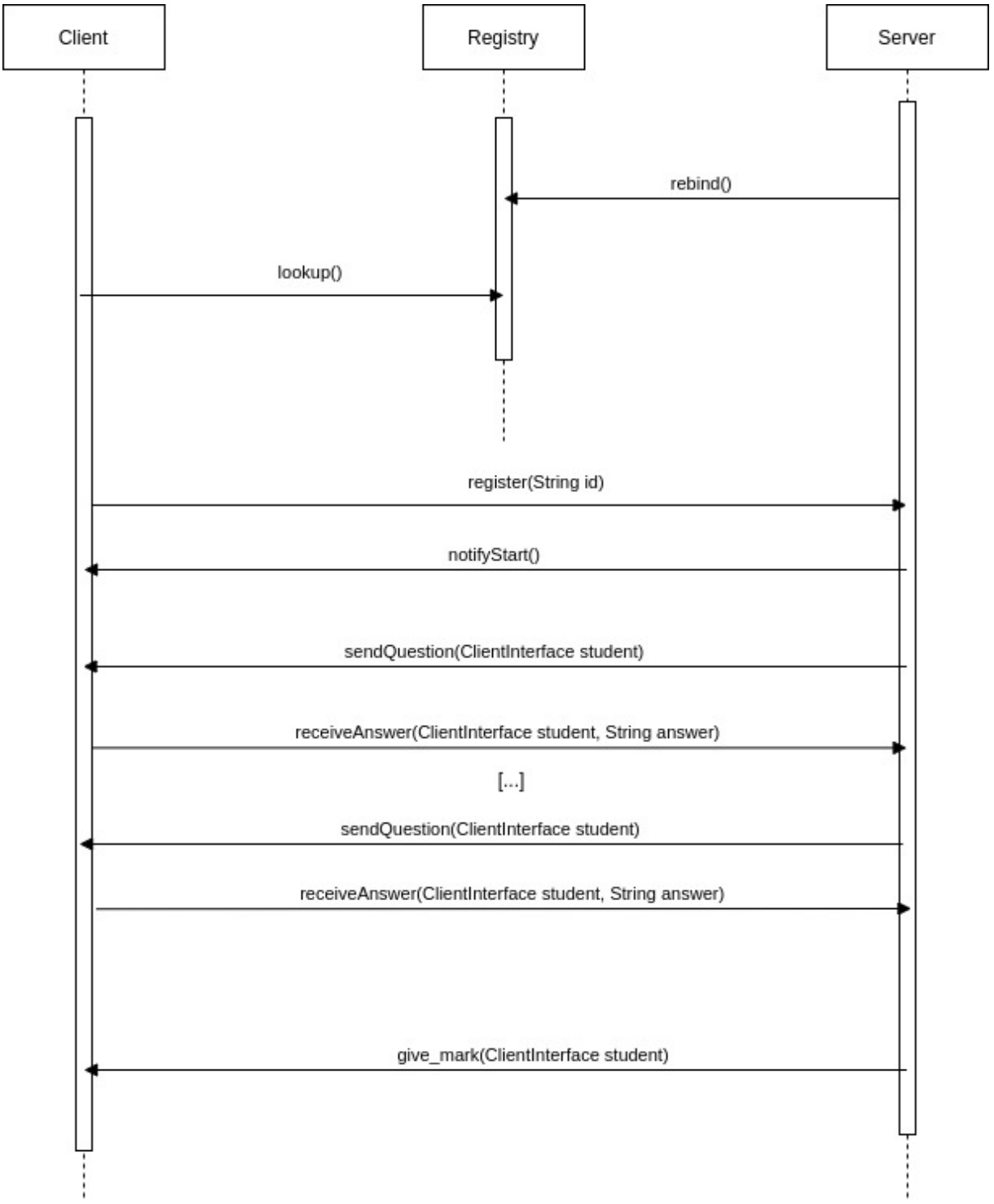
3.1 Main Changes

This is a list of the main decision we took in order to provide an adequate solution to the problem statement.

- Adding Input Handler for Server: in order to allow the teacher to interact with the server we would create a separate thread responsible to monitor the keyboard input, this way we avoid blocking the server to do so.
- Removing attributes from ClientImplementation: the server will be the one in charge to control all the information about the client, this way, the server doesn't depend on the client being up to access it.
- Creating Exam class: to simplify the amount of data structures stored in the server, we created a different class to store all the information related to the exam, as well as reading it from the csv file.
- SendQuestion from ClientInterface moved to ServerImplmentation: initially we intended that it would be the client the one in charge to ask to the server for the next question. In order to simplify the flow of the communication, now it's the server that sends it any time the student answers a question, without the need of requesting the next one.
- SendQuestion and SendChoices unified: we unified all message passing to the client into a unique method called sendMessage.

For the remaining part, we added some more methods that we didn't realise that we would need initially.

4 Sequence diagram



5 Client.java

This is the client that connects to the server. Basically it connects to the server, asks for the ID of the student (which is checked in order to avoid two students with the same ID). After that it waits for the exam to start, and then waits until the student ends or the exam does.

6 StudentImplementation.java

This class works as the student. It has the following methods:

- sendMessage() that processes the message for the client.
- notifyStart(). Prints that the exam has already began.
- denyConnection(). Sends the error that the connection was not possible because of some error. The error is entered as a parameter.
- give_mark(). When the student has ended or the exam has finished, this method shows the correct answers, the total number of questions and the score (correct/total * 10).

7 Exam.java

We've created a java class called Exam.java that stores the data of the exams in csv format. This class has the method read_csv that reads the csv file and saves the questions, choices and correct answer number in an object available to the server to access.
At the beginning, we though to implement this data structures inside the TeacherImplementation, but at the end we decided to implement this to keep everything sorted and simple.

8 Server.java

Server.java is the class that executes the server (aka the teacher). It has three important things to discuss:

- The startRegistry, which creates the Registry and start the comunications.
- The InputHandler class is on the end of the code but we'll explain it now to have later a better understanding of the main method. The InputHandler works as a thread that reads the screen inputs without blocking the server. InputHandler has two bool variables, the start and finish. When the exam begins, and the command "start" is entered, start becomes true and the server is notified. When the exam ends, entering "finish", the finish boolean turns true and the server gets notified.
- And we also have the main method, which begins by creating the background thread, using the InputHandler class. Then tries to connect to the registry, initialize the teacher (with TeacherImplementation) and then it waits for the students to connect. When the exam begins, it sends the first question to the students and then waits to be notified about the end of the exam. And when that moment comes, it gives the results to the clients, disconnects them and saves the scores on the file results.txt.

9 TeacherImplementation.java

This class works as the teacher that communicates with the clients through the server. This class has the following methods:

- notifyStart(). It notifies the students waiting that the exam has began.
- sendQuestion(). Sends the question, the possible answers, and asks a student for a valid answer. It saves the student progress inside the variable called progress.

- receiveAnswer(). This method checks if the student has ended the exam, answering all the questions, if the answer received is valid and if it's correct (and in that case adds points to the final mark).
- giveMark(). This is used when the student has answered all the questions in the exam. What it does is send the mark of that student to him.
- writeResults(). It creates the results.txt file and saves the marks of all students that registered before the exam began.