

Problem Description

The objective of this lab assignment is to gain practical experience in building a distributed software system using Java Remote Method Invocation (RMI). A distributed software system is a model where components on networked computers interact by exchanging messages to achieve a common goal. In this lab, you will enhance the Student Registration System from Lab 2 by transforming it into a distributed system using Java RMI.

You can use the RMI code sample on Canvas that was demoed during lecture or the following tutorial to learn more about Java RMI [Java RMI Tutorial](#)

Task

Your task is to turn the existing student registration system from Lab 2, including the required modifications, which are the logging, overbooking, and course-conflict checking components, into a distributed system. You will be provided with the architecture diagram below to help you develop a functional system.

Architecture of the Distributed System

Refer to Figure 1 for the architecture diagram of the current framework.

The system includes the following components:

- Student.java: Defines the Student class within the system.
- Course.java: Defines the Course class within the system.
- Courses.txt: A text file containing a list of courses.
- Students.txt: A text file containing a list of students.
- DBInterface.java: This is a Java interface that extends java.rmi.Remote. It defines the methods required for remote database operations, such as getAllStudentRecords() and makeARegistration(). These methods mirror the functionality of the original database object used in Lab 2. Instances of classes implementing this interface will be passed as arguments to the constructors of all handler classes during runtime.
- Database.java: This class is responsible for handling data access operations, specifically reading data from Courses.txt and Students.txt. It implements the DBInterface.java interface. Additionally, the class includes a main method, which serves as an entry point to instantiate the Database object, create a registry, and rebind the instance to that registry for remote access.
- Client.java: This client-side component is responsible for interfacing with remote objects based on user input. It incorporates a command line interface (CLI) that allows users to invoke specific remote methods. Please note that the printing and logging should be done on the client side.
- IActivity.java: This interface extends java.rmi.Remote and defines the execute() method, enabling remote execution of tasks. It is implemented by all the handler classes from Lab 2, such as

`ListStudents`. Each implementation encapsulates its specific logic—for example, listing all students—within the `execute()` method.

- `Server.java`: This server-side component is responsible for instantiating all handler objects from Lab 2 that implement the `IActivity` interface and binding them to a dedicated registry. Additionally, it performs a lookup to retrieve the `DBInterface` instance through a separate registry. This registry is created and initialized within the main method of the `Database.java` class. Notably, the registries for the handler objects and the Database are distinct entities, each operating on different port numbers.

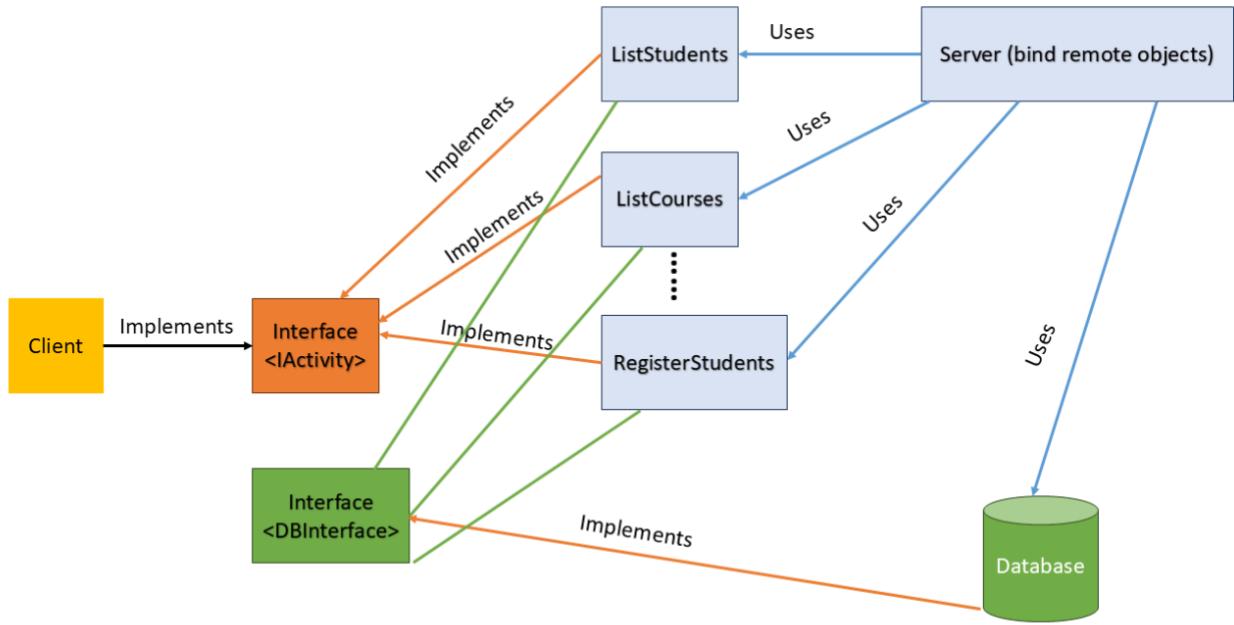


Figure 1. The architecture diagram of the distributed system

Grading Criteria

- The correct implementation and operation of the solutions - we will test your solutions with our test data.
- The degree to which your solutions adhere to the distributed software system where possible to do so.