Distributed Systems - COMP SCI 3012
Collaborative Session 1

Gia Bao Hoang - a1814824

Marcus Hoang - a1814303

I. Goal 1: Implementing Basic Server-Client Interaction

Goal: Create a server and client where the client will enter their name, and the server will register it and print out *"<Client name> is connecting to the server"*

Steps:

- **Define a Remote Interface:** Declare the remote methods that will be available to the client. In our program, we will have a method for registering the client's name.

- **Implement the Remote Interface:** Create a server-side class that provides concrete implementations of the methods declared in the remote interface.

- **Compile the Interface and Implementation:** Use the Java compiler **javac** to compile the remote interface and its server-side implementation.

- **Create a Registry for the Server:** Using the **rmiregistry** tool, we create a registry on the server machine. This is where the client will look up the remote object.

- **Start the Server:** The server will instantiate the remote object and bind it to the registry, making it available to clients.

- **Create the Client:** On the client side, we created a program that looks up the remote object in the registry and invokes the method to register the name.

- **Running the Programs:** Run both the client and server programs, making sure the RMI registry is running.

Diagram:

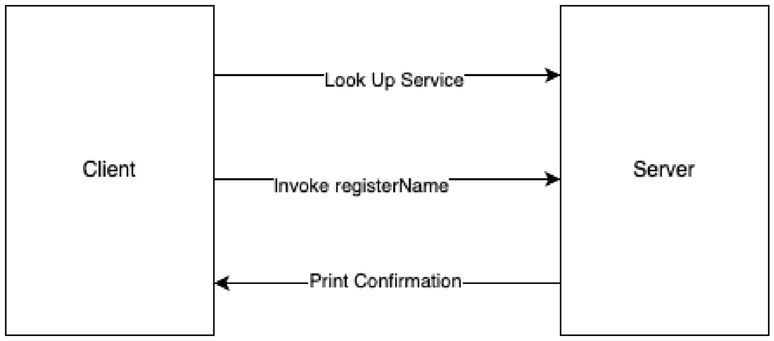

Diagram 1: the interaction between Client and Server in our program.

- **Client Looks Up the Remote Object in the Registry:** The client asks the server's registry for the remote object responsible for registering names.

- **Invoke registerName Method:** The client calls the registerName method on the remote object, sending the client's name as an argument.

- **Print Confirmation on Server:** The server prints the *"<Client name> is connecting to the server"* message.

II. Goal 2: Extending Server Functionality with Persistent State Management

Goal: Update the above program in Goal 1 by creating **adding** and **removing** client names in an array or list maintained on the server.

Steps:

- **Modify the Remote Interface:** Add methods to the remote interface to handle adding and removing client names.

- Update the Server Implementation: Modify the server implementation to maintain a list of client names. We updated the methods to add and remove names from the list.

- Compile the Updated Classes: We recompiled both the updated remote interface and the server implementation.

- Create a Registry for the Server: We reused the existing registry or create a new one.

- Start the Updated Server: We need to run the server with the updated implementation.

- Update the Client: Modify the client to call the new methods for adding and removing names.

- Running the Updated Programs: Run both the updated client and server programs, making sure the RMI registry is running.

Diagram:

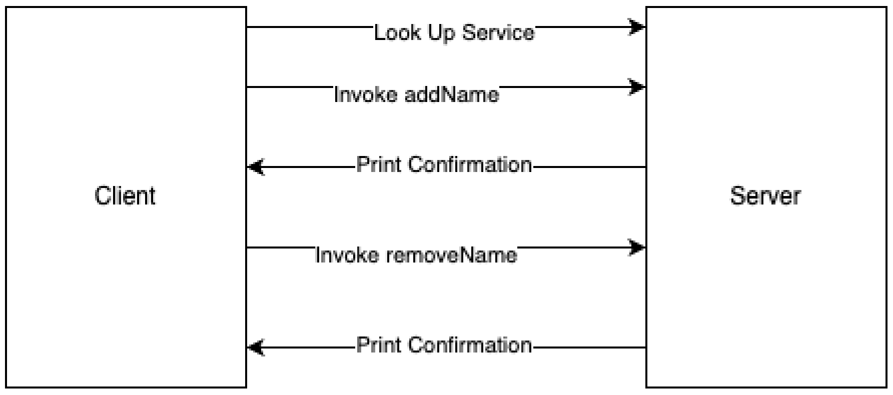

Diagram 2: the updated interaction between Client and Server in our program

- Client Looks Up the Remote Object in the Registry: The client asks the server's registry for the remote object responsible for registering names.

- Invoke addName Method: The client calls a new addName() method on the remote object, sending the client's name as an argument. The server adds the name to its list.

- Print Confirmation on Server: The server prints a message confirming the addition of the client's name.

- Invoke removeName Method: If needed, the client can call a removeName() method to remove its name from the server's list.

- Print Confirmation of Removal on Server: The server prints a message confirming the removal of the client's name.

III. Goal 3: Ensuring System Integrity and Scalability through Targeted Testing

Acceptance Testing

- **Goal & Rationale:** Validates that the system meets predefined requirements and user needs, ensuring the entire process works as intended.
- **Conduct:**
  + User Scenarios: Create scenarios that reflect end-user interactions.
  + Execution: Perform tests manually or through automated scripts.

        + Criteria & Reporting: Measure success against predefined outcomes and document results.

Unit Testing

- **Goal & Rationale:** Tests individual components to ensure each works as intended, providing a solid foundation for the entire system.
- **Conduct:**
    + Test Cases: Write tests for specific functions like adding and removing names.
    + Test Frameworks: Use tools like JUnit for structured testing.
    + Mocking & Continuous Integration: Utilize mocking for isolation and integrate into continuous development pipelines.

Performance Testing

- **Goal & Rationale:** Assesses how the system performs under load, crucial for understanding scalability and identifying potential future issues, especially as the number of users increases.
- **Conduct:**
    + Load Generation: Simulate multiple clients connecting simultaneously.
    + Metrics & Analysis: Measure response times, throughput, and resource utilization. Analyze to identify bottlenecks.
    + Stress Testing & Baseline Comparisons: Determine system limits and compare against standards to gauge performance.

Conclusion:

The combination of these three tests ensures:

- **Functional Correctness:** Through Acceptance and Unit Testing, the system's overall functionality and the correctness of individual components are verified.
- **Scalability Insight:** Performance Testing provides vital information about how the system can handle increased loads, preparing it for future growth.

Together, these testing types provide a comprehensive assessment tailored to the application's simplicity and potential scalability requirements.