Assignment No 1

Title: Implement Multi-threaded Client/Server Process Communication using RMI

Objectives

- To understand the principles of Remote Method Invocation (RMI) in Java.
- To develop a client-server model using Java RMI.
- To implement multi-threading on the server-side for handling concurrent client requests.
- To explore inter-process communication (IPC) through network-based distributed systems.
- To gain practical experience in building distributed applications in Java.

Problem Statement

Design and implement a Java-based distributed application that demonstrates process communication between a server and multiple clients using Remote Method Invocation (RMI). The server must be capable of handling multiple client requests simultaneously using multi-threading. Each client can request a remote method to perform a specific task (such as arithmetic operation, string manipulation, or database query), and the server should process these requests concurrently and return the results to the clients.

Expected Outcomes

- A functional Java RMI application with:
  - A remote interface defining the methods to be invoked by clients.
  - A server-side implementation that registers the remote object and handles concurrent requests using threads.
  - Multiple client programs that invoke remote methods and receive responses.
- Understanding of how RMI abstracts away low-level socket programming.
- Demonstrated capability of handling concurrent requests using multi-threading.
- Insight into distributed computing and client-server architecture.

Theory:

Remote Method Invocation (RMI) is a Java mechanism that enables an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. This is a form of **Inter-Process Communication (IPC)** that facilitates building distributed applications in Java.

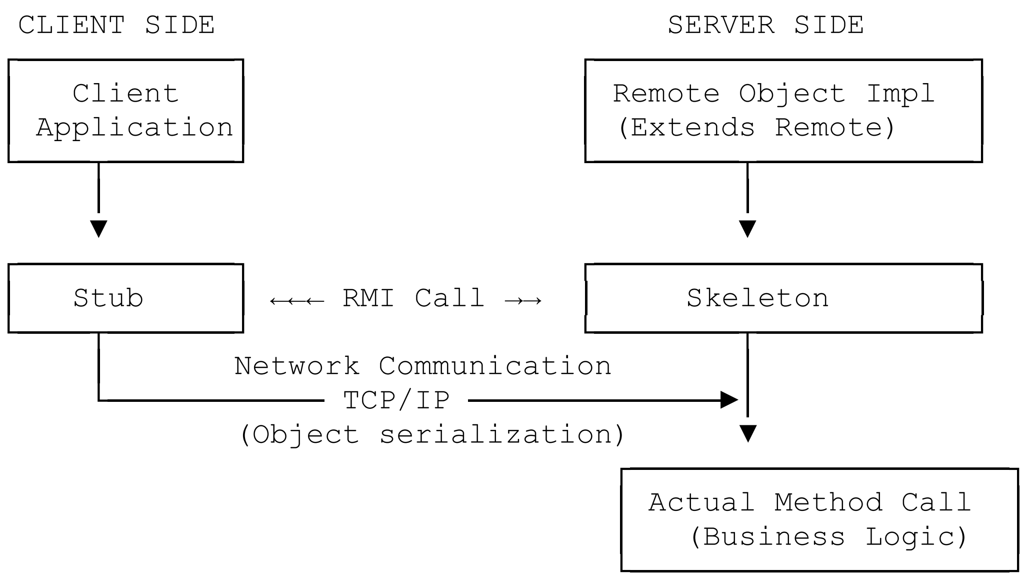
RMI abstracts the network communication, allowing developers to focus on invoking methods as if they were local. The system handles locating the remote object, passing the method call parameters, executing the method remotely, and returning the results.

Key Components of Java RMI

Component	Description
Remote Interface	Declares the methods that can be called remotely. It extends `java.rmi.Remote`.
Remote Object (Implementation)	Implements the remote interface and contains the logic. Extends `UnicastRemoteObject`.
Stub	A client-side proxy that represents the remote object. Used to invoke methods remotely.
Client	The application that looks up the remote object from the RMI Registry and calls methods.
RMI Registry	A simple name service that allows clients to get references to remote objects.
Skeleton (Pre-Java 5)	Server-side entity that dispatches calls to the actual remote object (now replaced internally by dynamic proxies).

RMI Architecture Diagram

Here's a simple visual representation of how Java RMI works:



Note: From Java 5 onwards, **Skeletons** are no longer explicitly required. Java dynamically handles method invocation using reflection and proxies.

Flow of Execution in Java RMI

1. **Define a Remote Interface**: The interface declares the methods that can be called remotely.
2. **Implement the Remote Interface**: A class implements this interface and extends `UnicastRemoteObject`.

3. **Start the RMI Registry**: A separate process (`rmiregistry`) is started to register remote objects.
4. **Bind Remote Object**: The server registers the remote object with a name in the RMI registry.
5. **Client Lookup**: The client contacts the registry to retrieve the reference (stub) of the remote object.
6. **Remote Method Invocation**: The client calls a method using the stub, which communicates over the network.
7. **Server Execution**: The server executes the requested method and returns the result to the client.

Multi-threading in RMI Server

- By default, Java RMI is **multi-threaded**: each client request is handled in its own thread.
- Developers can manage threads explicitly to enhance performance, implement synchronization, or limit concurrent client sessions.

Hardware Requirements

- **Processor**: Minimum Dual-core CPU
- **RAM**: 4 GB or more
- **Storage**: At least 100 MB of free disk space
- **Network**: Localhost or networked environment for testing multi-client communication.

Conclusion: