Relatório do Trabalho de Comunicação Cliente-Servidor via RMI

Maurício Miranda Carneiro - 548318
Maria Eduarda Santana Marques - *538264*

1. Objetivo do Projeto

O objetivo deste projeto foi reimplementar a primeira questão do trabalho utilizando a comunicação cliente-servidor via RMI (Remote Method Invocation), respeitando o protocolo de requisição-resposta descrito na seção 5.2 do livro texto. A solução implementada permite que o cliente envie requisições para o servidor e receba respostas, sem a utilização de sockets, uma vez que o RMI já se encarrega de toda a comunicação via rede.

A aplicação simula uma biblioteca remota, onde o cliente pode realizar operações como adicionar publicações, registrar clientes e consultar o acervo. A comunicação entre o cliente e o servidor foi organizada por meio de métodos remotos, e as mensagens de requisição e resposta foram empacotadas e tratadas conforme o protocolo especificado.

2. Estrutura do Sistema e Implementação

2.1 RMI - Remote Method Invocation

Para garantir a comunicação entre o cliente e o servidor, utilizei o RMI (Remote Method Invocation), que permite que o cliente invoque métodos no servidor como se estivessem na mesma máquina, apesar de estarem em sistemas distribuídos.

A implementação do RMI envolve duas partes essenciais:

- O servidor, que expõe os serviços remotos.
- O cliente, que consome esses serviços remotos.

2.2 Protocolo de Requisição-Resposta

O protocolo de requisição e resposta descrito na seção 5.2 do livro foi implementado da seguinte forma:

1. Método `doOperation`: O cliente envia uma requisição para o servidor, incluindo o nome do objeto remoto, o método a ser invocado e os argumentos. O servidor processa a requisição e retorna a resposta.

Assinatura:
```
public byte[] doOperation(RemoteObjectRef o, int methodId, byte[] arguments)
```

    -

2. Método `getRequest`: O servidor obtém a requisição de um cliente através de uma porta de servidor.

Assinatura:
```
public byte[] getRequest()
```

3. Método `sendReply`: O servidor envia a resposta de volta ao cliente, endereçando a resposta ao IP e porta do cliente.

Assinatura:
```
public void sendReply(byte[] reply, InetAddress clientHost, int clientPort)
```

3. Arquitetura e Design do Sistema

A arquitetura do sistema foi projetada para atender aos requisitos do trabalho, com a implementação de métodos remotos para comunicação, e o uso de JSON para a representação externa de dados.

3.1 Classes de Entidades

O sistema possui quatro classes de entidades que representam os dados principais da aplicação:

1. Publicacao (classe base para todas as publicações)
2. Livro (extende `Publicacao`)
3. Revista (extende `Publicacao`)
4. Cliente

Essas classes são usadas para representar os dados e realizar operações como adicionar publicações, remover publicações e registrar clientes.

3.2 Composição - Agregação e Extensão

- Agregação ("tem-um"): A classe `BibliotecaRemotaImpl` contém listas de objetos das classes `Publicacao` e `Cliente`. Ou seja, uma biblioteca "tem" publicações e clientes, mas uma publicação ou cliente pode existir independentemente da biblioteca.
- Extensão ("é-um"): A classe `Revista` e a classe `Livro` são especializações da classe `Publicacao`. Ambas estendem `Publicacao` e herdam suas propriedades e

métodos, além de adicionar atributos e métodos específicos.

3.3 Métodos Remotos

A aplicação implementa quatro métodos remotos essenciais para o funcionamento do servidor:

1. adicionarPublicacao(Publicacao p): Adiciona uma publicação ao acervo.
2. removerPublicacao(Publicacao p): Remove uma publicação do acervo.
3. buscarPorTitulo(String titulo): Busca publicações pelo título.
4. listarPublicacoes(): Lista todas as publicações da biblioteca.

Esses métodos foram expostos pelo servidor e invocados pelo cliente usando RMI.

4. Passagem de Parâmetros e Representação Externa de Dados

4.1 Passagem por Valor e Referência

A passagem por referência foi usada para os objetos remotos, como as publicações e os clientes. No caso do RMI, objetos são passados por referência entre o cliente e o servidor.

A passagem por valor foi utilizada quando os dados (como publicações) são enviados ao servidor em formato serializado (JSON) usando a classe `Serializador`.

4.2 Representação Externa de Dados (JSON)

Para representar os dados entre o cliente e o servidor, JSON foi escolhido como o formato de dados. Os objetos `Livro` e `Revista` são serializados para JSON antes de serem enviados ao servidor e desserializados ao serem recebidos. O uso de JSON atende à exigência de representação externa de dados, como especificado no trabalho.

A biblioteca Jackson foi utilizada para serialização e desserialização dos objetos, através da classe `Serializador`:

```
public class Serializador {
    private static final ObjectMapper mapper = new ObjectMapper();

    // Método para serializar objetos para JSON
    public static byte[] toJson(Object obj) throws IOException {
        return mapper.writeValueAsBytes(obj);
    }

    // Método para desserializar JSON para objetos
```

```
    public static <T> T fromJson(byte[] json, Class<T> type) throws
IOException {
        return mapper.readValue(json, type);
    }
}
```

5. Testes e Validações

A comunicação entre o cliente e o servidor foi testada utilizando os métodos adicionarPublicacao, registrarCliente, e infoPublicacao, além de uma consulta de publicações. Os testes confirmaram que a comunicação está funcionando corretamente, com as publicações sendo adicionadas, removidas e consultadas de acordo com as operações.

Observações Finais

- Ordem de Execução: O servidor deve ser iniciado antes do cliente.
- Configuração de Portas: Verifique se a porta 1100 está livre para o servidor RMI. Caso contrário, você pode configurar outra porta no código do servidor e cliente.
- Desempenho: O servidor está configurado para continuar em execução, aguardando novas conexões após o primeiro atendimento.