Relatório do Trabalho 2: Invocação Remota de Método (RMI)

Alunos: Calebe Sucupira e Josué Sucupira.

Modelagem do Serviço

Para atender aos requisitos da aplicação, foi modelado um serviço de loja virtual com as seguintes características:

Entidades (POJOs)

Foram criadas 7 entidades para representar os dados do sistema, atendendo ao requisito de no mínimo 4.

- **Produto (Classe Abstrata):** Classe base para todos os itens vendáveis.
- **Celular, Capa, Pelicula, PowerBank:** Classes concretas que herdam de Produto. Isso cumpre o requisito de no mínimo 2 composições por extensão ("é-um").
- **Vendedor:** Representa um vendedor do sistema.
- **Venda:** Representa uma transação de venda. Esta classe possui um objeto Vendedor e uma List<Produto>, cumprindo o requisito de no mínimo 2 composições por agregação ("tem-um").

Todas as entidades implementam a interface java.io.Serializable para permitir a passagem por valor entre cliente e servidor.

Interface de Serviço (ILojaService)

Foi definida uma interface para o contrato do serviço remoto, contendo 4 métodos, conforme o requisito.

- Produto consultarProduto(String codigo)
- String adicionarProduto(Produto p)
- List<Produto> listarEstoque()
- String realizarVenda(Venda v)

Arquitetura do Protocolo RMI

O protocolo de comunicação foi implementado para seguir o fluxo de requisição-resposta.

- **Fluxo de Requisição:** No lado do cliente, o método doOperation (na classe Requestor) é responsável por empacotar uma Mensagem e enviá-la ao servidor.
- **Fluxo de Resposta:** No lado do servidor, a classe ClientHandler (atuando como skeleton) possui a lógica para receber a requisição (getRequest) e enviar a resposta (sendReply).

Estrutura da Mensagem

A classe Mensagem foi criada para encapsular os dados da comunicação, contendo os campos definidos no enunciado:

- messageType: (int) 0 para Requisição, 1 para Resposta.
- requestId: (int) Um número para identificar a requisição.
- objectReference: (String) O nome do serviço remoto a ser invocado (ex: "LojaService").
- methodId: (String) O nome do método a ser executado.
- arguments: (byte[]) Os argumentos do método, serializados em JSON.

Empacotamento de Dados (Marshalling)

A classe Marshaller é responsável por serializar e desserializar os argumentos e resultados dos métodos. Para cumprir o requisito de representação externa de dados, foi utilizado o formato **JSON**.

Para lidar com o polimorfismo da classe abstrata Produto, foi utilizada a classe auxiliar RuntimeTypeAdapterFactory, fornecida pela equipe do Gson. Ela inspeciona os dados e instancia a subclasse correta (Celular, Capa, etc.) durante a desserialização, evitando erros de instanciação de classes abstratas.

Passagem de Parâmetros

O sistema implementa os dois tipos de passagem de parâmetros exigidos:

- **Passagem por Valor:** Quando um cliente envia um objeto (ex: um Produto para ser adicionado), o objeto é serializado em JSON, enviado pela rede, e uma nova cópia é criada no servidor. Isso cumpre o requisito de passagem por valor.
- **Passagem por Referência:** O cliente não possui uma cópia do objeto de serviço. Ele interage com um LojaServiceStub, que atua como um proxy (ou referência remota). As chamadas de método no stub são convertidas em mensagens que usam a objectReference para identificar o objeto único que existe no servidor, simulando a passagem por referência para o serviço remoto.

Execução e Testes

A aplicação é executada em modo texto, conforme permitido. Para testar o sistema:

1. Primeiro, executa-se a classe Servidor.java, que inicia o serviço e aguarda conexões.
2. Em seguida, executa-se a classe Cliente.java, que realiza uma sequência de chamadas a todos os métodos remotos.

Os logs impressos nos consoles do cliente e do servidor demonstram o fluxo de requisição e resposta e o sucesso de cada operação.