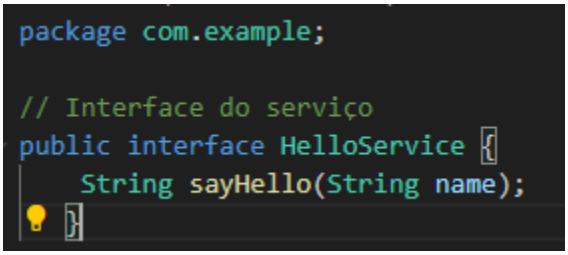
Este relatório apresenta a implementação de um sistema de **Remote Procedure Call (RPC)** utilizando **sockets** em Java. O objetivo é permitir que um cliente invoque um método remoto **(sayHello)** de um servidor, recebendo a resposta como se o método estivesse sendo executado localmente.

1. Interface **HelloService:**

   Essa interface define o contrato com o serviço remoto, garantido que qualquer implementação forneça o método **sayHello**.



```java
package com.example;

// Interface do serviço
public interface HelloService {
    String sayHello(String name);
}
```

2. Implementacao **HelloServiceImpl:**

   Fornece a implementação do método say Hello(), retornando uma mensagem personalizada;

   public class HelloServiceImpl implements HelloService: Define uma classe chamada **HelloServiceImpl,** que implementa a interface **HelloService**, isso significa que ele deve fornecer uma implementação para o método **sayHello;**

   @Override public String sayHello(String name): Este método recebe um nome como parâmetro e retorna uma mensagem personalizada.

```java
package com.example;

// Implementação do serviço
public class HelloServiceImpl implements HelloService {
    @Override
    public String sayHello(String name) {
        return "Olá, " + name + "! Este é um exemplo de RPC com sockets.";
    }
}
```

3. Servidor **Server.java**

Essa classe implementa um servidor socket que processa chamadas remotas usando um esquema simples de **RPC (Remote Procedure Call)** com comunicação baseada em strings

```
try (ServerSocket serverSocket = new ServerSocket(4000)) {
    System.out.println("Servidor pronto na porta 4000...");
```

.

O servidor é iniciado na porta 4000 esperando as conexões de clientes

```
while (true) {
    Socket clientSocket = serverSocket.accept();
    System.out.println("Cliente conectado!");
```

O servidor aceita conexões de clientes de forma contínua dentro de um **while(true).**

```
try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {
```

in lê as mensagens enviadas pelo cliente e **out** envia as mensagens de volta para o cliente.

```
String input = in.readLine();
System.out.println("Requisição recebida: " + input);
```

O servidor lê a requisição enviada pelo cliente.

```
if (input.startsWith("sayHello:")) {
    String name = input.split(":")[1];
    HelloService service = new HelloServiceImpl();
    String response = service.sayHello(name);
    // Envia a resposta de volta ao cliente
    out.println(response);
} else {
    out.println("Método não suportado.");
}
```

Se a requisição começar com **“sayHello:“**, o servidor **extrai o nome** e chama o método **sayHello()**.O resultado é enviado de volta ao cliente, se a requisição for inválida retorna “método não suportado“.

```
clientSocket.close();
```

Após processar a solicitação, o servidor fecha a conexão com o cliente.

4. Cliente **Client.java:**

Essa classe implementa um **cliente socket** que se comunica com um servidor **RPC (Remote Procedure Call)** simples, enviando uma solicitação e recebendo uma resposta.

```
try (Socket socket = new Socket("localhost", 4000);
```

O cliente cria uma conexão TCP com o servidor, que está rodando no endereço localhost na porta 4000.

```
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {
```

in: le as mensagens do servidor;

out: Envia a mensagem para o servidor. O parâmetro true ativa o auto flush, garantindo que os dados sejam enviados automaticamente.

```
String request = "sayHello:Usuário";
out.println(request);
System.out.println("Requisição enviada: " + request);
```

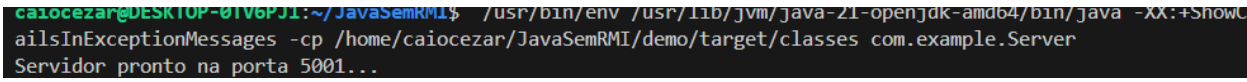
O cliente envia uma **string no formato "sayHello:Usuário"**, representando a chamada do método remoto **sayHello("Usuário")**.

```
String response = in.readLine();
System.out.println("Resposta do servidor: " + response);
```

O cliente aguarda uma resposta do servidor e a exibe no console.

5. Testes e resultados

5.1 Execução do servidor.



5.2 Execução do cliente.