

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
FLUMINENSE

FELIPE MUROS

**SISTEMAS DISTRIBUÍDOS: observações sobre a implementação de RMI na  
plataforma Microsoft® Windows 10 com as ferramentas Java e bibliotecas de  
java.rmi**

Relatório apresentado ao Instituto Federal de  
Educação, Ciência e Tecnologia Fluminense  
como parte das exigências para a conclusão da  
disciplina de Sistema Distribuídos do Curso  
Bacharelado em Sistemas de Informação.

Professora: Msc. [Maria Alciléia Alves Rocha](#)

Campos dos Goytacazes, RJ

Abril, 2022

## SUMÁRIO

<b>1. INTRODUÇÃO</b>	<b>3</b>
1.1 Recursos utilizados	3
<b>2. REVISÃO BIBLIOGRÁFICA</b>	<b>5</b>
2.1 java.net.MalformedURLException	5
2.2 java.rmi.registry.LocateRegistry	5
2.3 java.rmi.registry.Registry	5
2.4 java.rmi.Remote	5
2.5 java.rmi.server.Unicast	5
2.6 java.rmi.activation.Activatable	6
2.7 java.rmi.NotBoundException	6
2.8 java.rmi.RemoteException	6
2.9 java.rmi.server.ServerNotActiveException	6
<b>3. IMPLEMENTAÇÃO</b>	<b>7</b>
3.1 Programa principal Cliente	8
3.2 Interface comum ao Cliente e Servidor	9
3.3 Classe que implemente a interface no Servidor	9
3.4 Programa principal Servidor	11
3.5 Classe TempoDeOperacao no Servidor	11
<b>4 RESULTADOS</b>	<b>13</b>
<b>REFERÊNCIAS</b>	<b>16</b>

## ÍNDICE DE FIGURAS

Figura 1: Especificações do dispositivo, Fonte (Autor) .....	3
Figura 2: Especificações do Windows, Fonte (Autor) .....	4
Figura 3: Versão Apache NetBeans IDE, Fonte (Autor) .....	4
Figura 4: Estrutura dos pacotes Cliente e Servidor .....	7
Figura 5: Solicitação do cliente na rodada 1 .....	13
Figura 6: Resposta recebida na rodada 1 .....	13
Figura 7: Servidor rodando .....	13
Figura 8: Servidor respondendo na rodada 1 .....	14
Figura 9: Servidor respondendo na rodada 2 .....	14
Figura 10: Resposta recebida na rodada 2 .....	14
Figura 11: Finalização do programa do cliente .....	15

## 1. INTRODUÇÃO

O objetivo desse trabalho é pesquisar sobre a comunicação entre processos e desenvolver um programa para estabelecer a comunicação entre dois processos utilizando RMI com base na linguagem de programação Java.

Os resultados obtidos serão apresentados, bem como o código fonte desenvolvido com comentários para facilitar o entendimento do leitor.

### 1.1 Recursos utilizados

Computador tipo desktop:

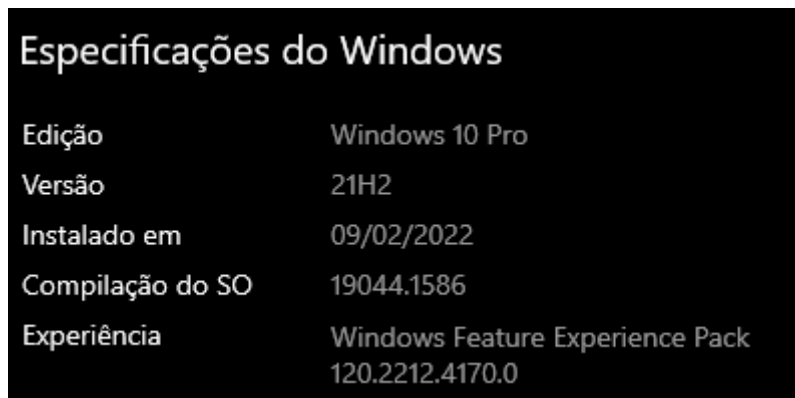
- Intel Core I7 12700F;
- 16GB RAM DDR 4 3200MHz;
- SSD Raid 0 1Tb,
- HDD Raid 10 500Gb 5400RPM;
- HDD 500Gb 5400RPM
- Geforce GTX 1080 8GB VRAM;

Conforme Figura 2.

Especificações do dispositivo	
Nome do dispositivo	DESKTOP-LHMP76G
Processador	11th Gen Intel(R) Core(TM) i7-11700F @ 2.50GHz 2.50 GHz
RAM instalada	16,0 GB (utilizável: 15,9 GB)
ID do dispositivo	BEA1ED94-3D17-4EB3-AED8-A1F8E6C7AAED
ID do Produto	00330-80000-00000-AA291
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Caneta e toque	Nenhuma entrada à caneta ou por toque disponível para este vídeo

Figura 1: Especificações do dispositivo, Fonte (Autor)

Especificações da versão do Windows conforme Figura 3.

A screenshot of the 'Especificações do Windows' (Windows Specifications) window. It has a dark background with white text. The title is 'Especificações do Windows'. Below it, there are five rows of information, each with a label on the left and a value on the right.

Edição	Windows 10 Pro
Versão	21H2
Instalado em	09/02/2022
Compilação do SO	19044.1586
Experiência	Windows Feature Experience Pack 120.2212.4170.0

Figura 2: Especificações do Windows, Fonte (Autor)

Programado na linguagem Java utilizando a IDE Net Beans, versão da IDE, Java e JSE conforme Figura 4

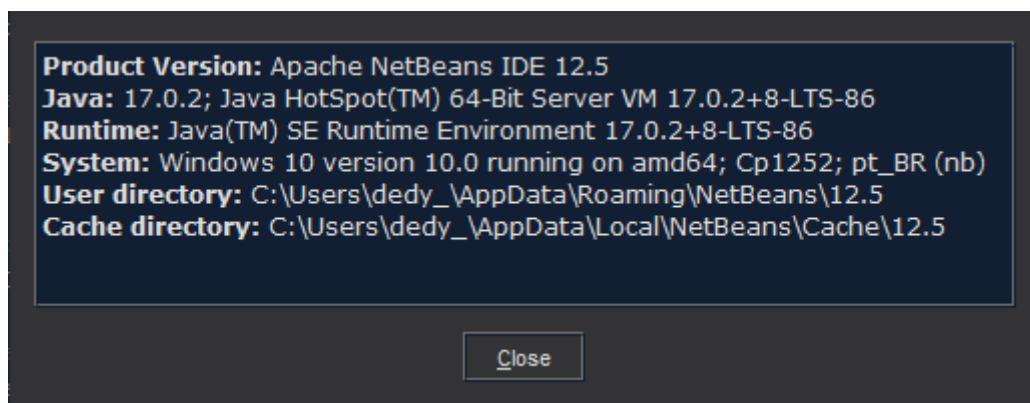


Figura 3: Versão Apache NetBeans IDE, Fonte (Autor)

## 2. REVISÃO BIBLIOGRÁFICA

Para este trabalho, além da linguagem java, foram utilizadas algumas bibliotecas para auxílio na programação com RMI De acordo Oracle (1993), a saber:

### 2.1 `java.net.MalformedURLException`

Essa classe lança uma exceção quando uma URL é informada de maneira incorreta.

### 2.2 `java.rmi.registry.LocateRegistry`

*LocateRegistry* é usado para obter uma referência a um registro de objeto remoto de *bootstrap* em um host específico (incluindo o host local) ou para criar um registro de objeto remoto que aceita chamadas em uma porta específica.

### 2.3 `java.rmi.registry.Registry`

*Registry* é uma interface remota para um registro de objeto remoto simples que fornece métodos para armazenar e recuperar referências de objeto remoto vinculadas a nomes de *string* arbitrários. Os métodos *bind*, *unbind* e *rebind* são usados para alterar as ligações de nomes no registro, e os métodos *lookup* e *list* são usados para consultar as ligações de nomes atuais.

### 2.4 `java.rmi.Remote`

A interface *Remote* serve para identificar interfaces cujos métodos podem ser invocados de uma máquina virtual não local. Qualquer objeto remoto deve implementar direta ou indiretamente essa interface.

### 2.5 `java.rmi.server.Unicast`

Usado para exportar um objeto remoto com JRMP e obter um *stub*<sup>1</sup> que se comunica com o objeto remoto. Os *stubs* são gerados em tempo de execução usando objetos proxy dinâmicos ou são gerados estaticamente em tempo de compilação,

normalmente usando a ferramenta `rmic`.

## 2.6 `java.rmi.activation.Activatable`

A classe *Activatable* fornece suporte para objetos remotos que requerem acesso persistente ao longo do tempo e que podem ser ativados pelo sistema.

As classes listadas a seguir tem suas definições de acordo com Oracle (1993) e de acordo com Plus (2021) são indispensáveis em métodos principais de manipulação de RMI.

## 2.7 `java.rmi.NotBoundException`

Lança uma exceção se for feita uma tentativa de pesquisar ou desvincular no registro um nome que não tenha sido associado anteriormente;

## 2.8 `java.rmi.RemoteException`

Superclasse comum para várias exceções relacionadas à comunicação que podem ocorrer durante a execução de uma chamada de método remoto. Cada método de uma interface remota, uma interface que estende *java.rmi.Remote*, deve listar *RemoteException* em sua cláusula *throws* (lançamento de exceção).

## 2.9 `java.rmi.server.ServerNotActiveException`

Lança uma exceção durante uma chamada para *RemoteServer.getClientHost* se o método *getClientHost* for chamado fora de uma chamada de método remoto em serviço.

### 3. IMPLEMENTAÇÃO

O código foi comentado para facilitar o entendimento, desta maneira foi dispensado um texto explicativo específico. No entanto é importante destacar que a interface é comum ao no lado do cliente e do servidor, pois de acordo com Plus (2003), para que a comunicação aconteça, o cliente precisa conhecer a interface que roda no servidor, assim ela precisa estar presente em ambos os lados.

Assim do lado do cliente existe a interface *FatInterface* (que descreve os métodos para manipulação de dados no servidor) e o programa principal (*main*) que fará a solicitação ao usuário de um número para o cálculo de seu fatorial, o programa envia ao servidor o número e recebe como resposta o resultado do cálculo, como o desafio do trabalho foi apenas obter o resultado do cálculo não foi utilizado o protocolo *Marshalling* para transferência de objetos.

Do lado do servidor existe a interface *FatInterface*, o programa principal (*main*) que “ouve” a rede para receber solicitação do servidor e devolver a resposta, além de instanciar um objeto que implementa a interface que calcula o fatorial, assim o cliente solicita deste objeto o procedimento de calcular fatorial e recebe o resultado do cálculo, por fim a classe *TempoDeOperacao* está presente no servidor apenas para ilustrar o funcionamento de programas com classes de conhecimento apenas do servidor e que auxiliam no processamento de uma requisição do cliente, esta classe é responsável por calcular o tempo de processamento da requisição do cliente. A estrutura dos pacotes Cliente e Servidor podem ser vistas na Figura 4.

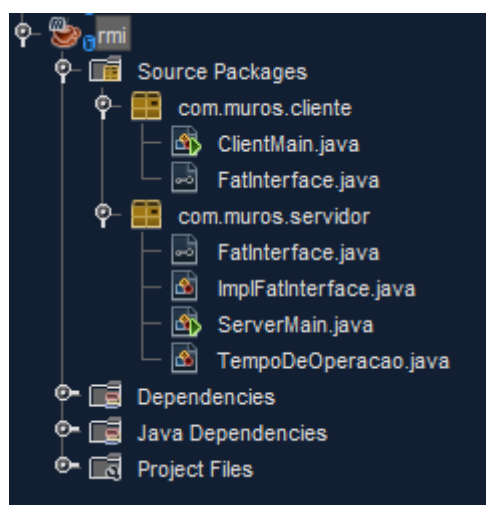


Figura 4: Estrutura dos pacotes Cliente e Servidor

### 3.1 Programa principal Cliente

```
package com.muros.cliente;

//importação das bibliotecas RMI
import java.net.MalformedURLException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import javax.swing.JOptionPane;
//importação da interface RMI para fatorial
import com.muros.servidor.FatInterface;

public class ClientMain {
    //cria um objeto de interface RMI que vai conter a referência do objeto remoto
    private static FatInterface objServerRef;

    public static void main(String[] args) throws MalformedURLException,
RemoteException, NotBoundException, ServerNotActiveException {
        Integer numero;
        while (true){
            try{
                //Buscar o registro no servidor especificado pelo IP 192.168.1.179
                Registry registro = LocateRegistry.getRegistry("192.168.1.179",
5005);

                //Buscar a referência do objeto exportado pelo servidor no Registro
do RMI nomeado como 'Fat'
                FatInterface refFat = (FatInterface) registro.lookup("Fat");

                //define a janela para pegar o número do fatorial
                numero = Integer.parseInt(JOptionPane.showInputDialog("Digite um
número para calcular o fatorial."
                    + "\nOu um número negativo para encerrar"));

                if (numero >= 0){
                    //chama o método calcular fatorial do objeto remoto, passando o
valor inserido pelo usuário.  exibe o resultado na tela
                    JOptionPane.showMessageDialog(null, "Fatorial de " + numero + "
= " + refFat.calcularFatorial(numero));
                }
                else{
                    System.out.println("App do cliente encerrado com sucesso\n");
                }
            }
        }
    }
}
```



```

        System.exit(0);
    }
}
catch (Exception e){
    System.err.println("Erro no cliente: "+e.toString());
    e.printStackTrace();
    System.exit(0);
}
}
}
}
}

```

### 3.2 Interface comum ao Cliente e Servidor

```

package com.muros.cliente;

//importação dos pacotes RMI necessários
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;
//define a interface
//determina que a interface estende de Remote, o que diz que ela pode ter seus
métodos invocados por uma máquina virtual não local
public interface FatInterface extends Remote{

//define o esqueleto do método calcular e obriga que ele implemente as exceções de
Remote e ServerNotActive
    int calcularFatorial(int numero) throws RemoteException,
ServerNotActiveException;
}

```

### 3.3 Classe que implemente a interface no Servidor

```

package com.muros.servidor;

//importação dos pacotes RMI
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;
import java.rmi.server.UnicastRemoteObject;

import com.muros.servidor.TempoDeOperacao;
import com.muros.servidor.FatInterface;

```

```

//define a classe servidor que implementa a interface remota Fatorial
public class ImplFatInterface extends UnicastRemoteObject implements FatInterface{
    private static final long serialVersionUID = 1L;

    //o método construtor do UnicastRemoteObject exporta o objeto para o RMI
    protected ImplFatInterface() throws RemoteException {
        super();
    }

    //implementa o corpo do código para calcular o fatorial
    //recebe o número passado pelo cliente
    public int calcularFatorial(int numero) throws RemoteException,
    ServerNotActiveException{
        System.err.println("O cliente de IP " + getClientHost() +
            ", solicitou o cálculo do fatorial de " + numero + "!\n");

        /*inicia um objeto para guardar a hora inicial da execução do fatorial
        apenas para mostrar que o servidor pode manipular outros objetos que não são
        retornados ao cliente,
        e são utilizados para manipular o solicitado*/
        TempoDeOperacao ObjCalcTempo = new TempoDeOperacao();

        //realiza o cálculo do fatorial
        int resultadoFatorial = this.fatorial(numero);

        //exibe o tempo que o cliente levou para executar o cálculo do fatorial
        System.out.println("O cliente de IP " + getClientHost() + " levou " +
            ObjCalcTempo.finish() + " ns para calcular o fatorial de " + numero +
            "!\n");

        //retorna o resultado para o cliente
        return resultadoFatorial;
    }
}

//define um método recursivo para calcular o fatorial do número passado
private int fatorial(int numero) {
    if(numero <= 1) {
        return 1;
    } else {
        return numero * fatorial(numero - 1);
    }
}
}

```

### 3.4 Programa principal Servidor

```
package com.muros.servidor;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ServerMain {

    public static void main(String[] args){
        try {
            //configura o nome do servidor como 192.168.1.179
            System.setProperty("java.rmi.server.hostname", " 192.168.1.179");

            //cria um objeto que possui o método de cálculo do fatorial
            ImplFatInterface objCalculofatorial = new ImplFatInterface();

            //Registrar o nome do objeto que será exportado pelo servidor e
            //requisitado pelo cliente
            Registry registro = LocateRegistry.createRegistry(5005);

            //faz o bind do stub nomeando como "Fat"
            registro.rebind("Fat",objCalculofatorial);

            System.out.println("O servidor está pronto para receber requisição de
            cliente");
        } catch (Exception e) {
            System.err.println("Erro no servidor: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

### 3.5 Classe TempoDeOperacao no Servidor

```
//Classe para cálculo de tempo, isolada da classe de Fatorial
package com.muros.servidor;

public class TempoDeOperacao {
    private final long startTime;

    public TempoDeOperacao() {
        this.startTime = System.nanoTime();
    }
}
```

```
        //retorna o tempo em nanosegundos levado para cálculo do fatorial (tempo do  
momento - inicial)  
        public long finish(){  
            return (System.nanoTime() - this.startTime);  
  
        }  
    }
```

## 4 RESULTADOS

Como resultado podemos observar o funcionamento correto do programa, A Figura 7 mostra a mensagem do servidor como pronto para receber a requisição do fornecedor, e como pode ser visto na Figura 5, o cliente solicitou o cálculo do fatorial de 5, recebendo resposta o valor 120 conforme visto na Figura 6, a Figura 8 mostra que o servidor identificou o cliente no endereço 192.168.1.142 e enviou a respostas conforme solicitação, processando o cálculo em 13500 nano segundos.

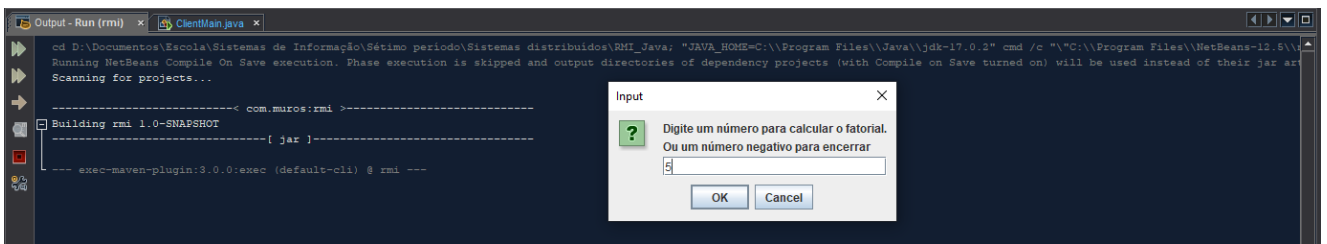


Figura 5: Solicitação do cliente na rodada 1

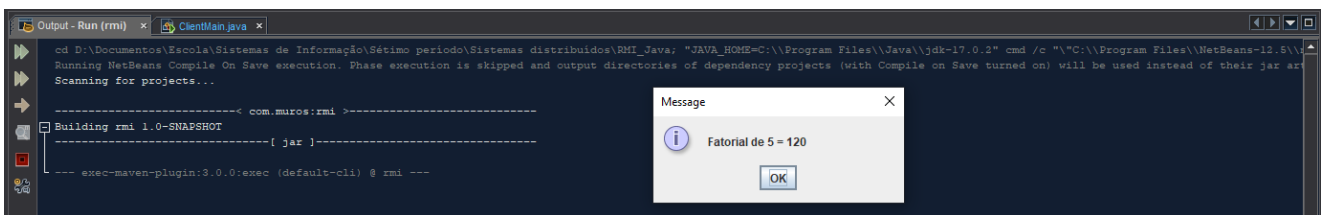


Figura 6: Resposta recebida na rodada 1

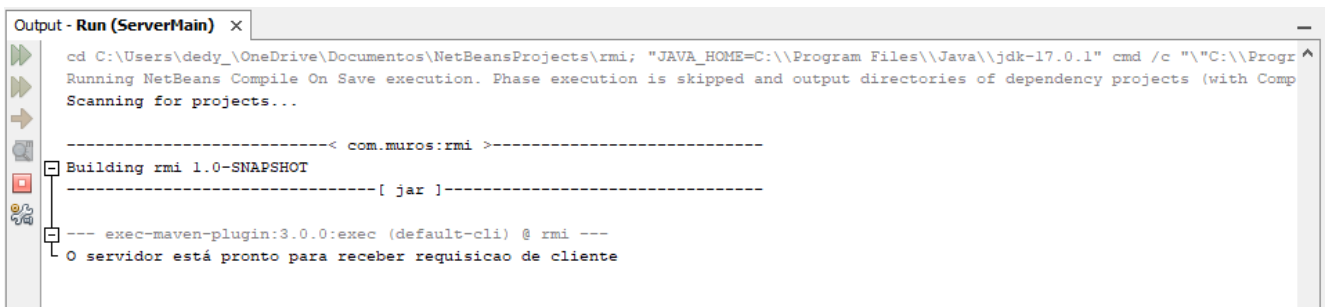
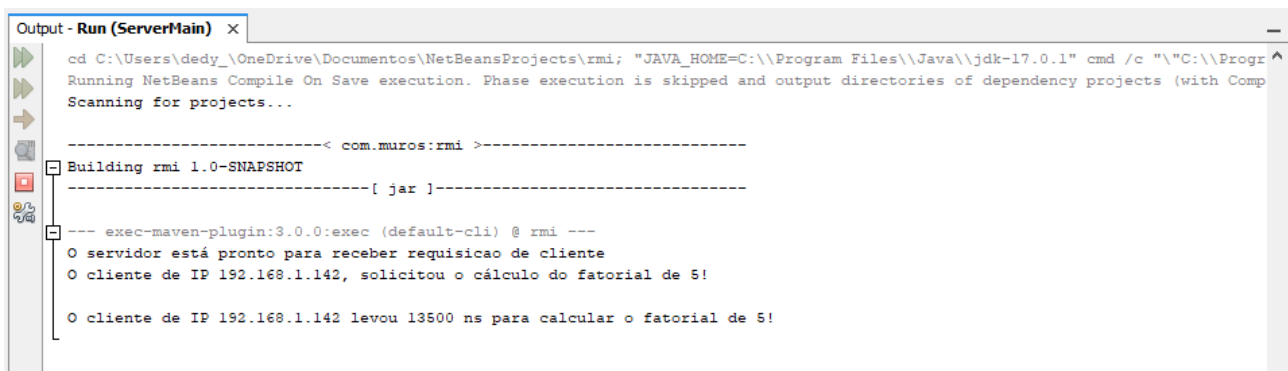


Figura 7: Servidor rodando



```
Output - Run (ServerMain) x
cd C:\Users\dedy_\OneDrive\Documentos\NetBeansProjects\rmi; "JAVA_HOME=C:\\Program Files\\Java\\jdk-17.0.1" cmd /c "%C:\\Progr
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Comp
Scanning for projects...

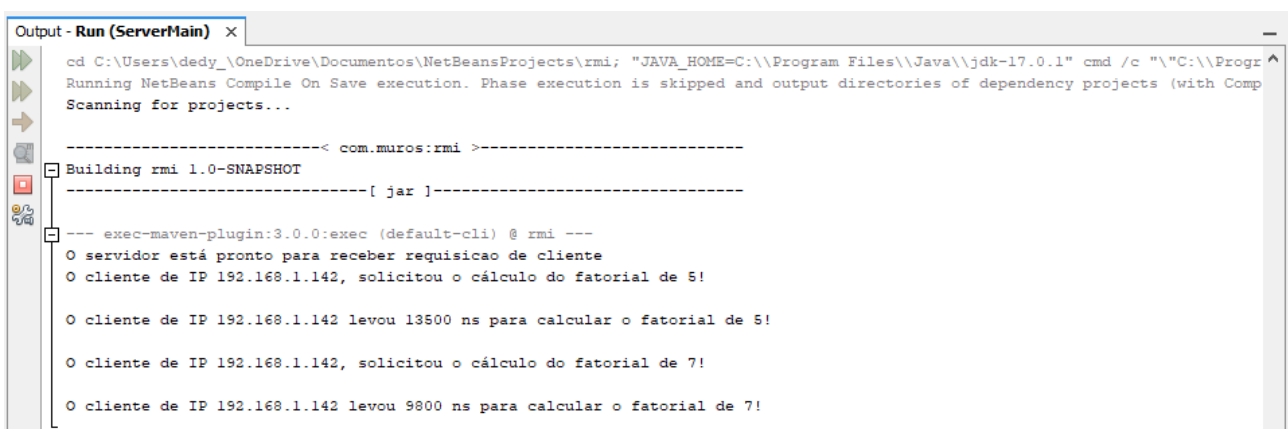
-----< com.muros:rmi >-----
Building rmi 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ rmi ---
O servidor está pronto para receber requisicao de cliente
O cliente de IP 192.168.1.142, solicitou o cálculo do fatorial de 5!

O cliente de IP 192.168.1.142 levou 13500 ns para calcular o fatorial de 5!
```

Figura 8: Servidor respondendo na rodada 1

Foi executada uma segunda rodada de cálculo com sem reiniciar o servidor a Figura 9 mostra o que o servidor realizou os cálculos em 9800 nano segundos, e mesmo após o cliente encerrar o aplicativo, o servidor permanece aguardando novas solicitações até que seja manualmente interrompido. A Figura 10 mostra a resposta que o cliente recebeu solicitando o fatorial de 7. A Figura 11 mostra que o programa encerrou do lado do cliente após o usuário entrar número negativo, condição para sair do loop e encerrar o programa.



```
Output - Run (ServerMain) x
cd C:\Users\dedy_\OneDrive\Documentos\NetBeansProjects\rmi; "JAVA_HOME=C:\\Program Files\\Java\\jdk-17.0.1" cmd /c "%C:\\Progr
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Comp
Scanning for projects...

-----< com.muros:rmi >-----
Building rmi 1.0-SNAPSHOT
-----[ jar ]-----

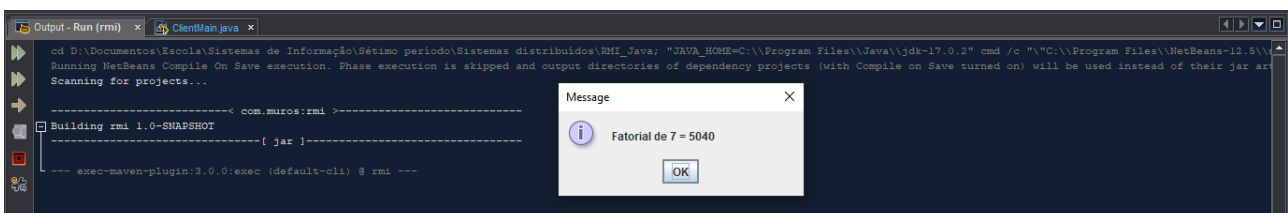
--- exec-maven-plugin:3.0.0:exec (default-cli) @ rmi ---
O servidor está pronto para receber requisicao de cliente
O cliente de IP 192.168.1.142, solicitou o cálculo do fatorial de 5!

O cliente de IP 192.168.1.142 levou 13500 ns para calcular o fatorial de 5!

O cliente de IP 192.168.1.142, solicitou o cálculo do fatorial de 7!

O cliente de IP 192.168.1.142 levou 9800 ns para calcular o fatorial de 7!
```

Figura 9: Servidor respondendo na rodada 2



```
Output - Run (rmi) x ClientMain.java x
cd D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\RMI_Java; "JAVA_HOME=C:\\Program Files\\Java\\jdk-17.0.1" cmd /c "%C:\\Program Files\\NetBeans-12.5\\
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Compile on Save turned on) will be used instead of their jar an
Scanning for projects...

-----< com.muros:rmi >-----
Building rmi 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ rmi ---
```

Message

Fatorial de 7 = 5040

OK

Figura 10: Resposta recebida na rodada 2

```
cd D:\Documentos\Escola\Sistemas de Informação\Sétimo período\Sistemas distribuídos\RMI_Java; "JAVA_HOME=C:\\Program Files\\Java\\jdk-17.0.2" cmd /c "%C:\\Program Files\\NetBeans-12.5\\...
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Compile on Save turned on) will be used instead of their jar an
Scanning for projects...

----- com.muros:rmi >-----
Building rmi 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ rmi ---
Digitado o número: -1.
App do cliente encerrado com sucesso

BUILD SUCCESS

Total time: 9.790 s
Finished at: 2022-04-11T22:14:18-03:00
```

Figura 11: Finalização do programa do cliente

## REFERÊNCIAS

ORACLE. **Java™ Platform, Standard Edition 7 - API Specification**. 1993. Disponível em <https://docs.oracle.com/javase/7/docs/api/java/rmi>. Acesso em: 09 abr. 2022.

ORACLE. **Class MalformedURLException**. 1993. Disponível em <https://docs.oracle.com/javase/8/docs/api/java/net/MalformedURLException.html>. Acesso em: 09 abr. 2022.

PLUS, Tutorial. **JAVA RMI programming tutorial**. Youtube, 11 de agosto de 2021. Disponível em: <https://www.youtube.com/watch?v=NmGytKDhCx4&t=1316s>. Acesso em: 09 abr. 2022.

SARMENTO, Luís. **Breve Introdução ao RMI - Remote Method Invocation**. 2003. Disponível em: <https://web.fe.up.pt/~eol/AIAD/aulas/JINIdocs/rmi1.html>. Acesso em: 11 abr. 2022.