APLICAÇÃO JAVA RMI DE AGENDA DE TELEFONES

Mathias Artur Schulz[1]
[1]Instituto Federal Catarinense – Campus Rio do Sul
mathiasschulz34@gmail.com

Abstract. *This article describes the development of a phone book application in the Java programming language using Java RMI (Remote Method Invocation). It presents the structure of the project, the classes necessary for the operation and example of application execution.*
Key-words: *Java; RMI; POO.*

Resumo. *Este artigo descreve o desenvolvimento de uma aplicação de agenda de telefones na linguagem de programação Java utilizando Java RMI (Remote Method Invocation). Apresenta a estrutura do projeto, as classes necessárias para o funcionamento e exemplo de execução da aplicação.*
Palavras-chave: *Java; RMI; POO.*

1. Introdução

O presente trabalho apresenta uma aplicação de agenda de telefone, no qual mantém um arquivo de nomes e números de telefones, desenvolvida na linguagem de programação Java.

A aplicação utiliza o mecanismo Java RMI (Remote Method Invocation), no qual permite a invocação de métodos que estão em diferentes máquinas virtuais Java (JVM) e podem estar em diferentes máquinas.

O cliente pode adicionar um telefone na agenda de telefones e listar os telefones já cadastrados, a partir de implementações realizadas no servidor e disponibilizadas pelo servidor para acesso.

A figura 1 abaixo apresenta a estrutura desenvolvida, na pasta *phonebook* se encontra as classes e a interface criadas para desenvolvimento do algoritmo e o arquivo txt chamado *data* responsável por salvar a agenda de telefones. Na pasta *classapp* se encontra o código Java compilado e pronto para ser executado.

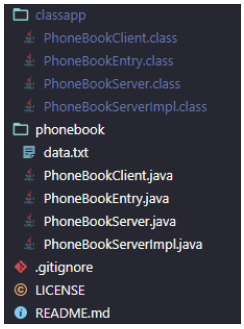

Figura 1 – Estrutura da aplicação

A figura 2 abaixo apresenta a estrutura do arquivo *data.txt* responsável por salvar todos os telefones e nomes da agenda. Cada telefone da agenda possui o nome, sobrenome e o número do telefone do dono, separados pelo símbolo "&". Além disso, cada telefone da agenda é separado com a quebra de linha (\n).

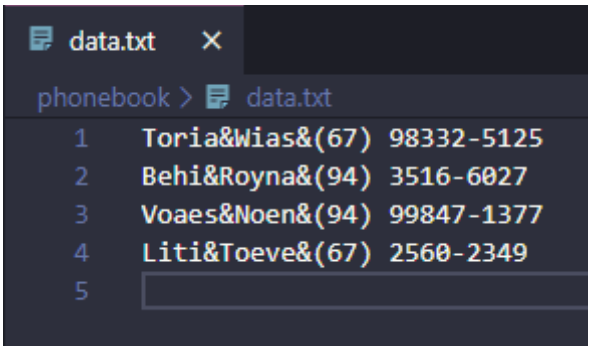

Figura 2 – Estrutura da agenda

Para compilar a aplicação, na pasta *java-rmi* do projeto é necessário executar o seguinte comando: "*javac -d ./ phonebook/*.java*", o projeto compilado estará salvo na pasta *classapp*. Após compilar o projeto, para executar a aplicação, na pasta *java-rmi* do projeto é necessário executar os seguintes comandos: Comando para executar o servidor: "*java classapp.PhoneBookServerImpl*"; Comando para executar o cliente "*java classapp.PhoneBookClient*". Com isso, as informações enviadas pelo cliente estarão salvas no arquivo *data.txt*.

2. Código documentado

Esta seção apresenta as classes e interface Java desenvolvidas para aplicação Java RMI de Agenda de Telefones.

A classe *PhoneBookEntry* da tabela abaixo apresenta a estrutura de um telefone da agenda. Possui os métodos privados de nome, sobrenome e telefone de um registro da agenda. Possui um construtor utilizado no momento da instanciação do objeto, para inicializar seus atributos de forma organizada, possui os métodos *getters* e *setters* para que esses atributos possam ser modificados e visualizados, apresenta um método *toString* para visualização organizada do objeto. Além disso, a classe também implementa a interface *Serializable*, para que o objeto possa ser armazenado e reconstruído posteriormente no mesmo ou em outro ambiente computacional.

```
1   package classapp;
2
3   import java.io.Serializable;
4
5   public class PhoneBookEntry implements Serializable {
6
7           private static final long serialVersionUID = 2290645081309697371L;
8
9           private String name;
10          private String lastname;
11          private String phone;
12
13          public PhoneBookEntry(String name, String lastname, String phone) {
14                  this.name = name;
```

```
15                  this.lastname = lastname;
16                  this.phone = phone;
17          }
18
19          public String getName() {
20                  return name;
21          }
22
23          public void setName(String name) {
24                  this.name = name;
25          }
26
27          public String getLastname() {
28                  return lastname;
29          }
30
31          public void setLastname(String lastname) {
32                  this.lastname = lastname;
33          }
34
35          public String getPhone() {
36                  return phone;
37          }
38
39          public void setPhone(String phone) {
40                  this.phone = phone;
41          }
42
43          public String toString() {
44                  StringBuilder builder = new StringBuilder();
45                  builder.append("PhoneBookEntry [name=");
46                  builder.append(name);
47                  builder.append(", lastname=");
48                  builder.append(lastname);
49                  builder.append(", phone=");
50                  builder.append(phone);
51                  builder.append("]");
52                  return builder.toString();
53          }
54  }
```

Tabela 1 – Classe PhoneBookEntry

A interface *PhoneBookServer* da tabela abaixo é utilizada para especificar o comportamento que as classes que a implementarem devem possuir, neste caso devem possuir os métodos *getPhoneBook*, que retorna uma lista com todos os telefones da agenda e *addEntry* que realiza o cadastro de um telefone na agenda. Esta classe também estende a interface *Remote* e todos os métodos assinados especificam a classe RemoteException na cláusula throws para garantir a robustez das aplicações no sistema RMI.

```
1   package classapp;
2
3   import java.rmi.Remote;
4   import java.rmi.RemoteException;
5   import java.util.ArrayList;
6
7   public interface PhoneBookServer extends Remote {
8
9       public ArrayList<PhoneBookEntry> getPhoneBook() throws RemoteException;
10      public void addEntry(PhoneBookEntry entry) throws RemoteException;
11
12  }
```

Tabela 2 – Interface PhoneBookServer

A classe *PhoneBookServerImpl* da tabela abaixo é a classe do servidor, que estende a classe *UnicastRemoteObject*, que possui alguns métodos necessários para o servidor e implementa a interface *PhoneBookServer* da tabela 2. Essa classe possui um construtor que apenas chama o construtor da classe pai *UnicastRemoteObject*. Possui o método *main*, utilizado para criar um objeto que implementa a interface *PhoneBookServer* e que é registrado como um servidor no registro do RMI, com o nome "PhoneBook", com isso pode ser localizado pelos clientes.

Esta classe também possui a implementação dos métodos da interface, que são:
– *getPhoneBook*: Esse método realiza a leitura do arquivo *data.txt*, armazena o arquivo no *Scanner* para que possa ser lido linha por linha até o seu fim. Em cada linha é utilizado o método *generateObject* para converter a *String* para um objeto *PhoneBookEntry*, após a conversão o objeto é adicionado na lista. No fim do método a lista é retornada;
– *addEntry*: Esse método recebe um objeto *PhoneBookEntry* como parâmetro, realiza a leitura do arquivo *data.txt* e escreve a *string* do objeto no arquivo. A conversão do objeto para uma *string* é realizada a partir do método *generateData*.

```
1   package classapp;
2
3   import java.io.BufferedWriter;
4   import java.io.FileWriter;
5   import java.io.IOException;
6   import java.io.File;
7   import java.io.FileNotFoundException;
8   import java.util.Scanner;
9   import java.rmi.AlreadyBoundException;
10  import java.rmi.RemoteException;
11  import java.rmi.registry.LocateRegistry;
12  import java.rmi.registry.Registry;
13  import java.rmi.server.UnicastRemoteObject;
14  import java.util.ArrayList;
15
16  public class PhoneBookServerImpl extends UnicastRemoteObject implements PhoneBookServer {
17
18          private final static String FILE_DIRECTORY = "/home/matt/Workspace/java-
19  rmi/phonebook";
20          private final static String FILE_NAME = "data.txt";
21      private static final long serialVersionUID = 1L;
22
23      public PhoneBookServerImpl() throws RemoteException {
24                  super();
25          }
26
27      public static void main(String[] args) {
28          try {
29              // Instancia o objeto servidor e o seu stub
30          PhoneBookServerImpl server = new PhoneBookServerImpl();
31
32              // Registra o stub para que possa ser obtido pelos clientes
33              Registry registry = LocateRegistry.createRegistry(5099);
34              registry.bind("PhoneBook", server);
35
36              System.out.println("Servidor pronto");
37          } catch (RemoteException | AlreadyBoundException ex) {
38              System.err.println(ex);
39          }
40      }
41
42      /**
43       * Retorna uma lista com todos os objetos salvos
44       *
45       * @throws RemoteException
```

```
46      */
47         public ArrayList<PhoneBookEntry> getPhoneBook() throws RemoteException {
48                 ArrayList<PhoneBookEntry> listPhones = new ArrayList<PhoneBookEntry>();
49                 try {
50
51                         File file = new File(FILE_DIRECTORY, FILE_NAME);
52                         Scanner s = new Scanner(file);
53
54                         while (s.hasNextLine()) {
55                                 String data = s.nextLine();
56                                 PhoneBookEntry phone = generateObject(data);
57                                 listPhones.add(phone);
58                         }
59                         s.close();
60                 } catch (FileNotFoundException e) {
61                         System.out.println("An error occurred.");
62                         e.printStackTrace();
63                 }
64                 return listPhones;
65         }
66
67         /**
68          * Adiciona um objeto no data.txt
69          *
70      * @param entry
71      * @throws RemoteException
72      */
73         public void addEntry(PhoneBookEntry entry) throws RemoteException {
74                 try {
75                  FileWriter writer = new FileWriter(new File(FILE_DIRECTORY, FILE_NAME),
76 true);
77             BufferedWriter bufferedWriter = new BufferedWriter(writer);
78
79             bufferedWriter.write(generateData(entry));
80                         bufferedWriter.close();
81         } catch (IOException e) {
82             e.printStackTrace();
83         }
84         }
85
86         /**
87          * Gera uma string do objeto para ser salvo no data.txt
88          *
89          * @param phone
90          * @return
91          */
92         private static String generateData(PhoneBookEntry phone) {
93                 return   phone.getName()   +   "&"   +   phone.getLastname()   +   "&"   +
94 phone.getPhone() + "\n";
95         }
96
97         /**
98          * Gera o objeto a partir de uma string salva no data.txt
99          *
100         * @param stringPhone
101         * @return
102         */
103        private static PhoneBookEntry generateObject(String stringPhone) {
104                String[] fields = stringPhone.split("&");
105                PhoneBookEntry phone = new PhoneBookEntry(
106                        fields[0], fields[1], fields[2]
107                );
108                return phone;
109        }
110 }
```

Tabela 3 – Classe PhoneBookServerImpl

A classe *PhoneBookClient* é a classe do cliente, no qual é obtido o stub do servidor a partir do nome setado no servidor, neste caso "*PhoneBook*". Com isso, o

cliente possui uma referência remota para o servidor no registro RMI e possui acesso aos métodos *addEntry* e *getPhoneBook*.

Neste caso, o cliente criou três objetos *PhoneBookEntry* e salvou cada um deles na agenda a partir do método "*addEntry"*. Após o cadastro, o cliente chamou o método "*getPhoneBook*" que retornou uma lista com todos os telefones cadastrados na agenda e, com isso, o cliente apresentou todos os telefones da agenda.

```
1   package classapp;
2
3   import java.net.MalformedURLException;
4   import java.rmi.Naming;
5   import java.util.ArrayList;
6   import java.rmi.NotBoundException;
7   import java.rmi.RemoteException;
8
9   public class PhoneBookClient {
10
11      public static void main(String[] args) throws MalformedURLException {
12
13          try {
14              // Obtêm o stub do servidor
15              PhoneBookServer stub = (PhoneBookServer)
16              Naming.lookup("rmi://localhost:5099/PhoneBook");
17
18              // Chama os métodos do servidor e apresenta os resultados
19              PhoneBookEntry phone1 = new PhoneBookEntry(
20                  "Mathias", "Schulz", "(83) 99553-1521"
21              );
22              stub.addEntry(phone1);
23              PhoneBookEntry phone2 = new PhoneBookEntry(
24                  "Fyoli", "Doar", "(83) 2695-0469"
25              );
26              stub.addEntry(phone2);
27              PhoneBookEntry phone3 = new PhoneBookEntry(
28                  "Peazu", "Mibau", "(55) 2551-1089"
29              );
30              stub.addEntry(phone3);
31
32              ArrayList<PhoneBookEntry> listPhoneBook = stub.getPhoneBook();
33
34              System.out.println("Lista de felefones cadastrados: ");
35              for (PhoneBookEntry phoneBookEntry : listPhoneBook) {
36                  System.out.println(phoneBookEntry.toString());
37              }
38          } catch (RemoteException | NotBoundException ex) {
39              System.err.println(ex);
40          }
41      }
42  }
```

Tabela 4 – Classe PhoneBookClient

3. Teste e validação da aplicação

Nesta seção é apresentado um exemplo de funcionamento do projeto. Na figura abaixo é apresentado o arquivo *data.txt*, com quatro telefones cadastrados e o terminal superior com o servidor rodando a aguardando o cliente.

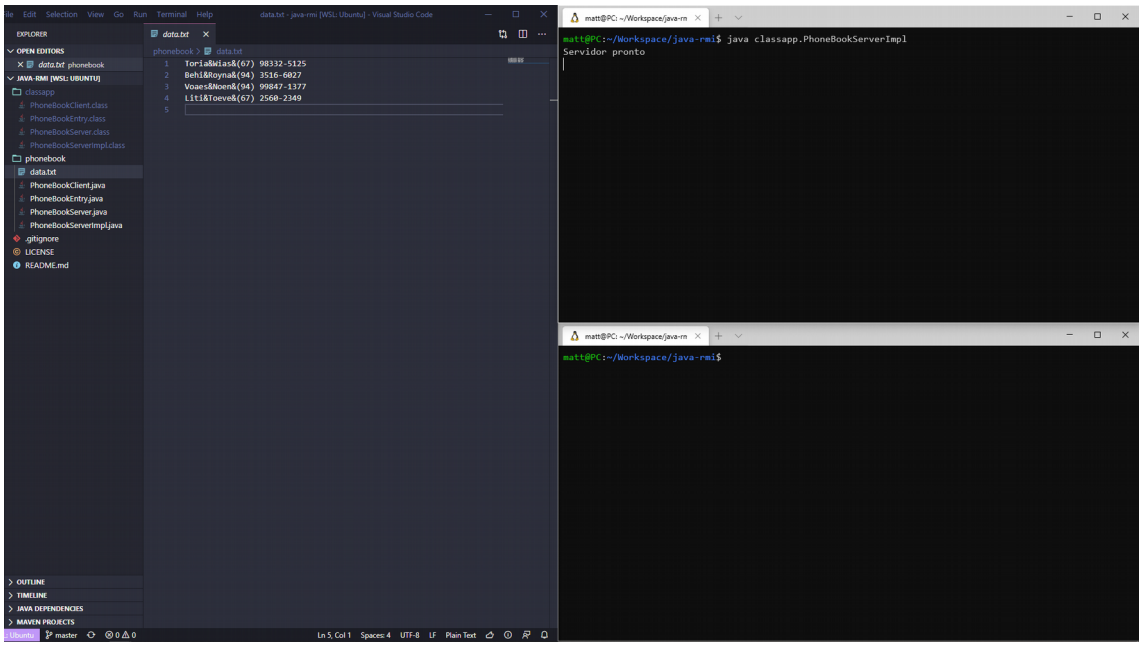

Figura 3 – Execução do servidor

Na figura abaixo é apresentado o arquivo *data.txt*, com sete telefones cadastrados, o terminal superior com o servidor rodando e o terminal inferior com o cliente já efetuado sua conexão com o servidor, no qual o cliente cadastrou três telefones e realizou a leitura da lista de telefones cadastrados. (O código pode ser observado na tabela 4).

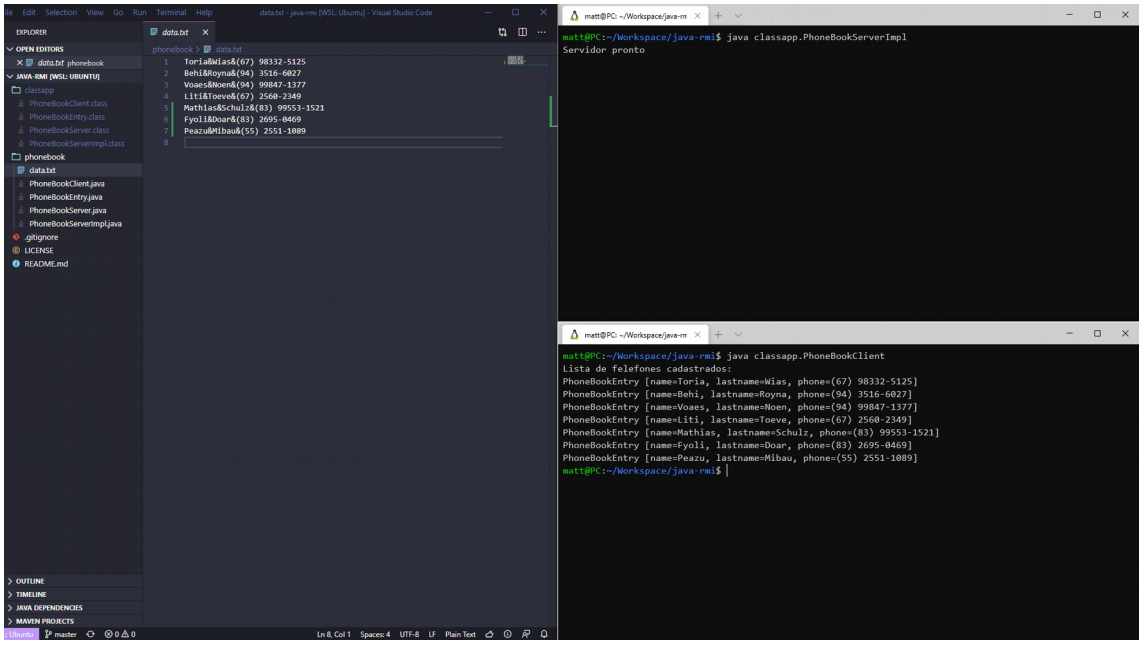

Figura 4 – Execução do cliente

O código para uma melhor visualização e realização de testes se encontra disponível no GitHub a partir do link: "https://github.com/mathiasarturschulz/java-rmi".