UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
SISTEMAS DISTRIBUIDOS

22/05/2019

Francisco Wesley Pedro Lima 398261

1 O Trabalho

A ideia principal do trabalho foi implementar um servidor que execute as funções de CRUD (Create, Delete, Remove e Update) utilizando Métodos de Invocação Remota (**RMI** - do Inglês **Remote Method Invocation**). Esse CRUD foi feito sobre uma lista de pessoas, que representam um banco de dados de cadastro de clientes.

Para implementar a ideia e um bom funcionamento do CRUD pelo menos duranto o tempo de execução, foi usado a estrutura de dados Vector do Java, uma vez que a implementação de um banco de dados não era exatamente necessária e tornaria o processo de criação apenas mais custoso e trabalhoso.

2 Divisões

O trabalho é divido, como de se esperar, apenas em um lado **Cliente** e um lado **Servidor**. Do lado Cliente temos apenas uma classe chamada *Client* que é reposável por todas as interações com o usuário, bem comoa conexão por meio de RMI com o servidor e o tratamento das informações que forem necessárias.

Do lado Servidor, temos as classes *Server*, *Pessoa*, *ShapeListServant* e *ShapeServant*. Temos tambéms as seguintes interfaces *ShapeList* e *Shape*. Tudo isso será melhor descrito a seguir.

3 Descrição das Classes

Desde já, vale salientar que o este documento não tem como propósito mostrar todo o código com as implementações básicas, mas sim, apenas algumas partes principais do mesmos e uma breve descrição. Portanto, haverá arquivos que o seu código não estará totalmente contido aqui.

3.1 Lado Cliente

3.1.1 Client

Essa classe, como descrita acima, contém basicamente a parte necessária para que o programa possa executar e as invocações remotas para que haja a comunicação com o lado servidor. Esse código executará enquanto o usuário não der o comando para que o mesmo pare. A seguir, veremos o código e logo em seguida, uma breve descrição de alguns trechos do mesmo

```
1  package T03RMI;
2
3  import java.awt.BorderLayout;
```

```java
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.util.Scanner;
import java.util.Vector;

public class Client {

    public static void main(String[] args) throws RemoteException {
        Scanner entrada = new Scanner(System.in);
        boolean control = true;
        int option;

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        } else {
            System.out.println("Already has a security manager, so cant
                set RMI");
        }
        ShapeList aShapeList = null;

        try {
            aShapeList = (ShapeList) Naming.lookup("//localhost/CRUD");
            while (true) {
                System.out.println("\n\n");
                System.out.println("Entre com uma opcao:");
                System.out.println("[1] Cadastrar pessoas");
                System.out.println("[2] Listar pessoas");
                System.out.println("[3] Atualizar pessoa");
                System.out.println("[4] Apagar pessoa");
                System.out.println("[5] Sair");
                option = Integer.parseInt(entrada.nextLine());
                System.out.println();

                Vector clientList = aShapeList.listar();

                switch (option) {
                    case 1:
                        String nome,
                         cpf,
                         email,
                         telefone;
                        System.out.println("Entre com o nome: ");
                        nome = entrada.nextLine();
                        System.out.println("Entre com o cpf: ");
                        cpf = entrada.nextLine();
                        System.out.println("Entre com o email: ");
                        email = entrada.nextLine();
                        System.out.println("Entre com o telefone: ");
                        telefone = entrada.nextLine();

```

```
53                          Pessoa pessoa = new Pessoa(nome, cpf, email,
                                telefone);
54                          System.out.println("Objeto pessoa criada!");
55                          if(aShapeList.cadastrar(pessoa)){
56                              System.out.println("Pessoa cadastrada com
                                    sucesso!");
57                              break;
58                          }
59                          System.out.println("Nao foi possivel cadastrar
                                ,\n"
60                                  + "CPF ja consta no sistema!");
61                          break;
62
63                      case 2:
64                          if(clientList.isEmpty()){
65                              System.out.println("Sem pessoas cadastradas
                                    !");
66                              break;
67                          }
68                          for (int i = 0; i < clientList.size(); i++) {
69                              Pessoa p = ((Shape) clientList.elementAt(i)
                                    ).getAll();
70                              p.print();
71                          }
72                          break;
73
74                      case 3:
75                          System.out.println("Entre com o CPF da pessoa a
                                ser alterada: ");
76                          cpf = entrada.nextLine();
77                          Pessoa atualizarRetornoPessoa = new Pessoa();
78                          atualizarRetornoPessoa = aShapeList.consultar(
                                cpf);
79                          if (atualizarRetornoPessoa != null) {
80                              System.out.println("Entre com o nome: ");
81                              nome = entrada.nextLine();
82                              System.out.println("Entre com o email: ");
83                              email = entrada.nextLine();
84                              System.out.println("Entre com o telefone: "
                                    );
85                              telefone = entrada.nextLine();
86
87                              Pessoa pessoaAtualizar = new Pessoa(nome,
                                    cpf, email, telefone);
88                              boolean retornoAtualizar = aShapeList.
                                    atualizar(pessoaAtualizar);
89
90                              if (retornoAtualizar) {
91                                  System.out.println("Pessoa atualizada
                                        com sucesso!");
```

```
                                } else {
                                    System.out.println("Erro, nao foi
                                        possivel atualizar a pessoa.");
                                }
                                break;
                            }

                            System.out.println("Pessoa nao encontrada!");
                            break;

                        case 4:
                            System.out.println("Entre com o CPF da pessoa a
                                ser apagada: ");
                            cpf = entrada.nextLine();
                            Pessoa deletarRetornoPessoa = new Pessoa();
                            deletarRetornoPessoa = aShapeList.consultar(cpf
                                );
                            if (deletarRetornoPessoa == null) {
                                System.out.println("Pessoa nao consta no
                                    sistema!");
                            } else {
                                boolean retornoApagar;
                                Pessoa pessoaApagar = new Pessoa(cpf);
                                retornoApagar = aShapeList.apagar(
                                    pessoaApagar);

                                if (retornoApagar) {
                                    System.out.println("Pessoa apagada com
                                        sucesso!");
                                    break;
                                }

                                System.out.println("Erro, nao foi possivel
                                    apagar a pessoa.");
                            }

                            break;

                        case 5:
                            System.out.println("Bye");
                            System.exit(0);
                            break;
                        default:
                            System.out.println("Opcao invalida");
                    }
                }

            } catch (RemoteException e) {
                System.out.println("All people: " + e.getMessage());
            } catch (Exception e) {
```

```
135                 System.out.println("Lookup: " + e.getMessage());
136                 e.printStackTrace();
137             }
138         }
139 }
```

Das linhas de 16 a 20 vemos a possíbilidade de estabelecer-mos uma coexão RMI por meio de uma conexão segura. Caso seja possível, a instanciamos na linha 24 e da linha 25 a linha 138 teremos o código que será responsável por executar enquanto o usário não finalizar a execução manualmente ou enquanto não ocorra uma exceção com relação a conexão.

Temos de forma explicita todos os passos referentes a coletagem de informações e conexão por meio do RMI para que o CRUD seja executado como esperado.

3.2 Lado Servidor

3.2.1 Server

```
1  package T03RMI;
2
3  import java.rmi.*;
4  import java.rmi.server.UnicastRemoteObject;
5
6  public class Server {
7
8      public static void main(String[] args) {
9          System.setSecurityManager(new SecurityManager());
10         try {
11             ShapeList aShapeList = new ShapeListServant();
12             ShapeList teste = (ShapeList) UnicastRemoteObject.exportObject(
                   aShapeList, 0);
13             Naming.rebind("CRUD", aShapeList);
14             System.out.println("Servidor de CRUD esta rodando");
15         } catch (Exception e) {
16             System.out.println("CRD server main " + e.getMessage());
17         }
18     }
19 }
```

Essa classe é responsável por aceitar a conexão por meio do RMI. A mesma cria objetos do tipo ShapeList que são as responsáveis por receber as requisições e tratar de acordo. Na linha 13 temos a nomeação do nosso serviço remoto que se chamará CRUD e o tipo de Objetos que estaremos aptos a receber.

3.2.2 Pessoa

```
1  package T03RMI;
2
3  import java.io.Serializable;
4
5  public class Pessoa implements Serializable {
6
```

```
7     private String nome;
8     private String cpf;
9     private String email;
10    private String telefone;
11
12    public Pessoa() {
13    }
14
15    public Pessoa(String cpf) {
16        this.cpf = cpf;
17    }
18
19    public Pessoa(String nome, String cpf, String email, String telefone) {
20        this.nome = nome;
21        this.cpf = cpf;
22        this.email = email;
23        this.telefone = telefone;
24    }
25
26    public void setNome(String nome) {
27        this.nome = nome;
28    }
29
30    public void setCpf(String cpf) {
31        this.cpf = cpf;
32    }
33
34    public void setEmail(String email) {
35        this.email = email;
36    }
37
38    public void setTelefone(String telefone) {
39        this.telefone = telefone;
40    }
41
42    public String getNome() {
43        return nome;
44    }
45
46    public String getCpf() {
47        return cpf;
48    }
49
50    public String getEmail() {
51        return email;
52    }
53
54    public String getTelefone() {
55        return telefone;
56    }
```

```
57
58      public void print() {
59          System.out.println("Nome = " + getNome() + ",\nCPF = " + getCpf() +
                ",\n"
60                  + "Email = " + getEmail() + ",\nTelefone = " + getTelefone
                        () + "\n");
61      }
62
63  }
```

A classe pessoa é responsável por instaciar os atributos que fazem parte do nosso serviço. Aqui temos operações que correspondem ao que deve ser especificado em uma classe do tipo modelo dentro de um padrão MVC (Model, Controller, View). Temos um método especial que é o **void print()**, responsável por retornar todas as informações de um dado objeto concatenado em formato String pronto para a impressão na tela.

3.2.3 ShapeList

```
1   public interface ShapeList extends Remote {
2
3       boolean cadastrar(Pessoa pessoa) throws RemoteException;
4
5       Vector listar() throws RemoteException;
6
7       Pessoa consultar(String cpf) throws RemoteException;
8
9       boolean atualizar(Pessoa pessoa) throws RemoteException;
10
11      boolean apagar(Pessoa pessoa) throws RemoteException;
12  }
```

Essa é uma interface que contém a assinatura a serem implementados e que correspondem a todas as operações que o CRUD deve realizar. Temos ainda na linha 7 a declaração de um método **consulta(String cpf)** que será usado por baixo dos panos apenas para fazer busca por uma pessoa com determinado CPF a fim de realizar alguma operação logo depois.

3.2.4 ShapeListServant

```
1   public class ShapeListServant implements ShapeList {
2       private Vector clientList;
3
4       public ShapeListServant(){
5           clientList = new Vector();
6       }
7
8       public boolean cadastrar(Pessoa pessoa) throws RemoteException {
9           if(consultar(pessoa.getCpf()) == null){
10              Shape s = new ShapeServant(pessoa);
11              clientList.addElement(s);
12              return true;
13          }
14          return false;
```

```
15      }
16
17      public Vector listar() throws RemoteException {
18          return clientList;
19      }
20
21      public Pessoa consultar(String cpf) throws RemoteException {
22          Pessoa pessoa = new Pessoa(cpf);
23          for (int i = 0; i < clientList.size(); i++) {
24              Pessoa p = ((Shape) clientList.elementAt(i)).getAll();
25              if (p.getCpf().equals(cpf)) {
26                  return p;
27              }
28          }
29          return null;
30      }
31
32      public boolean atualizar(Pessoa pessoa) throws RemoteException {
33          for (int i = 0; i < clientList.size(); i++) {
34              Pessoa p = ((Shape) clientList.elementAt(i)).getAll();
35              if (p.getCpf().equals(pessoa.getCpf())) {
36                  p.setNome(pessoa.getNome());
37                  p.setEmail(pessoa.getEmail());
38                  p.setTelefone(pessoa.getTelefone());
39                  return true;
40              }
41          }
42          return false;
43      }
44
45      public boolean apagar(Pessoa pessoa) throws RemoteException {
46          for (int i = 0; i < clientList.size(); i++) {
47              Pessoa p = ((Shape) clientList.elementAt(i)).getAll();
48              if (p.getCpf().equals(pessoa.getCpf())) {
49                  clientList.remove(i);
50                  return true;
51              }
52          }
53          return false;
54      }
55  }
```

A classe acima implementa a interface ShapeList, passando então a ter a obrigação de implementar todos os métodos antes instanciados.

Das linhas 8 a 15 temos o código responsável por cadastrar uma pessoa, ou seja, adicioná-la ao vector. É importante salietar que essa pessoa só será adicionada caso o seu CPF não conste já cadastrado.

Das linhas 17 a 19 temos o método responsável por retornar a lista com todas as pessoas cadastradas até então para o cliente.

Entre as linhas 21 e 30 temos a implementação do método de cosulta, já descrito acima.

Das linhas 32 a 43, temos a parte do código responsável por atualizar os dados de uma pessoa. O

referido método irá receber um novo objeto do tipo Pessoa e a partir daí, irá procurar uma pessoa já cadastrada no sistema que contenha o CPF repassado e alterar os seus dados.

Por fim, das linhas 45 a 54 encontramos a implementação do apagar, onde dado uma pessoa, o método vai verificar a existencia do mesmo no vector e removê-lo.

3.2.5 Shape

```
1  public interface Shape extends Remote {
2      Pessoa getAll() throws RemoteException;
3  }
```

Essa interface contém apenas a assinatura de um método que é responsável por retornar todos os dados do objeto pessoa.

3.2.6 ShapeServante

```
1  public class ShapeServant extends UnicastRemoteObject implements Shape {
2
3      Pessoa pessoa;
4
5      public ShapeServant(Pessoa p) throws RemoteException {
6          pessoa = p;
7      }
8
9      public Pessoa getAll() throws RemoteException {
10         return pessoa;
11     }
12 }
```

Além do seu construtor, essa classe, que implementa a interface Shape, contém a implementação do método responsável por retornar o objeto do tipo Pessoa quando estamos percorrendo o vector de pessoas a fim de fazer alguma modificação. Portanto, a sua função é apenas retornar o objeto com todos os seus dados de uma dada posição do vector.