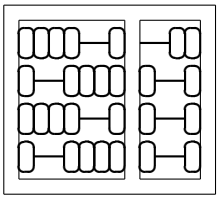
UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE COMPUTAÇÃO



SERVIDOR DE AGENDA BASEADO EM SOCKET UDP

Relatório do segundo laboratório de MC823

Aluno: Marcelo Keith Matsumoto **RA**: 085937

Aluno: Tiago Chedraoui Silva **RA**: 082941

Resumo

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

Sumário

1 Objetivo

O objetivoterceiro projeto de laboratório de teleprocessamento e redes é comparar duas implementações distintas do modelo cliente-servidor: Java RMI (Remote Method Invocation) e socket TCP. É de suma importância que utilizando a tecnologia Java RMI, cria-se uma agenda, para possibilitar uma comparação com a mesma agenda em socket TCP, criada anteriormente no projeto 1.

1.1 Teoria

Java RMI é uma das abordagens da tecnologia Java, construída para prover as funcionalidade de uma plataforma de objetos distribuídos e com sua API (Application Programming Interface) especificada pelo pacote java.rmi e seus subpacotes. A arquitetura RMI viabiliza a interação de um objeto ativo em uma máquina virtual Java com objetos de outras máquinas virtuais Java.

Aplicações que utilizam objetos distribuídos precisam das realizar as seguintes ações:

Localização de objetos remotos aplicações podem usar vários métodos para obter referências a objetos remotos. Ex: utilizar RMI registry

Comunicação com objetos remotos Detalhes da comunicação entre objeto remotos são gerenciados pelo RMI, ou seja para o programador chamadas remotas são similares a chamadas de métodos.

Carregamento de definições de classes para objetos móveis RMI prove mecanismos para carregar a definição de classes para um objeto assim como para transmitir seus dados

Para o desenvolvimento de uma aplicação cliente-servidor em Java RMI, são necessários um cliente e um para o servidor e a execução do serviço de registro de RMI (RMI registry)(Ver figura figura 1). Um servidor, em geral, instancia objetos remotos, referencia estes objetos e liga-os em uma determinada porta através de um bind, aguardando nesta porta os clientes invocarem os métodos destes objetos. Um cliente, em geral, referência remotamente um ou mais objetos remotos de um servidor, e invoca os métodos destes objetos. Os mecanismos pelos quais o cliente e o servidor se comunicam e trocam dados são fornecidos pelo Java RMI. O serviço de registro de RMI é uma implementação de um serviço de nomes para RMI, no qual cada serviço disponibilizado na plataforma é registrado através de um nome de serviço, ou seja, uma string única para cada objeto o qual implementa serviços em RMI.

2 Servidor de agenda

Para criar uma aplicação distribuída usando a tecnologia RMI deve-se projetar e implementar as componentes da aplicação. Primeiro define as interfaces,depois, baseado nas interfaces, implementa-se os objetos e posteriormente o cliente.

O sistema implementado, uma agenda distribuída, se baseia numa comunicação cliente-servidor. Nele o servidor possui todas as informações da agenda que estão armazenadas em um banco de dados, assim como as opções de interações com os dados que são apresentadas aos clientes em formas de um menu. O cliente só escolhe alguma opção de interação com os dados de acordo com menu.

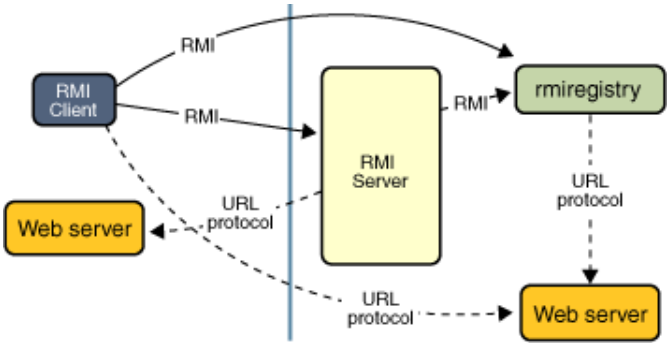


Figura 1: Aplicação distribuida RMI que usa o RMI registry para obter a referência para um objeto remoto.

2.1 Menu inicial

No menu inicial pode-se:

- Logar
- Criar novo usuário
- Sair

2.1.1 Login

O servidor pede ao usuário o nome de usuário, caso o nome estiver no banco de dados ele pede uma senha que é comparada ao valor do banco de dados, se o usuário não existir é avisado sobre a inexistência, se a senha não conferir é avisado que a senha não confere, caso contrário o usuário consegue logar no sistema, e o servidor recupera sua agenda (cada usuário possui sua agenda).

2.1.2 Novo usuário

O servidor pede um nome de usuário, o servidor verifica se o nome já não existe, se não existir pede a senha e armazena o usuário no sistema, assim como cria uma agenda vazia para o mesmo.

2.2 Menu usuário

Dentre as possibilidades de interações para um usuário logado tem-se:

- Inserção de um compromisso que possui um nome, dia, hora, e minuto.
- Remoção de um compromisso através de seu nome
- Pesquisa de compromisso por dia
- Pesquisa de compromisso por dia e hora
- Ver todos os compromisso de mês de abril

2.2.1 Inserção de compromisso

O usuário deve fornecer o nome do compromisso, o dia, a hora e o minutos em que ele ocorrerá. Caso o compromisso seja possível de ser alocado o servidor avisa com um "OK", se não for possível também é avisado de tal impossibilidade. Um compromisso é inserido ordenado na agenda se não existir um compromisso com mesmo horário.

2.2.2 Remoção de compromisso

O usuário deve fornecer o nome do compromisso que deve ser removido. Caso o compromisso seja encontrado ele é removido, caso contrário é dito que tal compromisso não existe. Se existirem dois compromissos de mesmo nome, o primeiro é removido. Logo é esperado que compromissos possuam nomes diferentes.

2.2.3 Pesquisas

O servidor faz um requerimento interativo, ou seja, se for selecionado a pesquisa por dia e hora, o servidor pergunta primeiramente o dia e depois a hora. Logo, é uma pesquisa em etapas no qual o servidor interage com nosso usuário.

3 Ambiente de implementação

O sistema de agenda foi implementado e executado nos seguintes sistemas operacionais :

- FC14 - Fedora Laughlin Linux 2.6.35.11
- Mac OS X 10.6.7

O sistema de agenda foi implementado na linguagem Java, utilizando a tecnologia RMI. Para o armazenamento dos dados, utilizou-se arquivos. Cada usuário possui um arquivo, a sua agenda, no qual armazena-se o nome do compromisso, o dia, a hora e o minuto do mesmo. O sistema lê a agenda a cada função chamada o servidor atualiza as informações dos arquivos.

O nosso sistema, além disso,apresenta transparência ao usuário. Os tipos de transparência a serem destacados são:

Acesso: Esconde as diferenças nas representações de dados e na invocação de funções ou métodos para facilitar a comunicação entre objetos da rede.

Localização: Esconde o local em que o objeto se encontra.

Concorrência: Esconde como as atividades são coordenadas entre os objetos para obter consistência em um nível mais alto.

4 Tempos de comunicação e total

Aplicamos o cálculo de tempo ao programa principal de forma a obtermos o tempo total, tempo de comunicação e os tempos da execução de cada função. Para o tempo total, no cliente pega-se o tempo antes da chamada da função do servidor e após o retorno dessa função.

Para o tempo de comunicação, pega-se o tempo total e subtrai-se o tempo de processamento do servidor, que é depois da chamada da função e antes do retorno da função. Para o tempo total das funções obteu-se o tempo de inserir um compromisso, remover o compromisso, ver a agenda do mês, ver a agenda de um dia, ver a agenda de uma hora. Os dados e os testes estão exemplificados nas tabelas seguintes:

Valor	Tempo
Max	3.902 ms
Min	2.933 ms
Mdia	3.318 ms
Desvio	0.145 ms

(a) Tempo total

Valor	Tempo
Max	3.385 ms
Min	2.592 ms
Mdia	2.870 ms
Desvio	0.105 ms

(b) Tempo de comunição

Tabela I: Inserir compromisso

Valor	Tempo
Max	15.788 ms
Min	11.238 ms
Mdia	12.063 ms
Desvio	0.171 ms

(a) Tempo total

Valor	Tempo
Max	3.370 ms
Min	2.551 ms
Mdia	2.878 ms
Desvio	0.052 ms

(b) Tempo de comunição

Tabela II: Remover compromissos

Valor	Tempo
Max	11.722 ms
Min	2.109 ms
Mdia	3.989 ms
Desvio	0.197 ms

(a) Tempo total

Valor	Tempo
Max	10.886 ms
Min	1.429 ms
Mdia	3.158 ms
Desvio	0.190 ms

(b) Tempo de comunição

Tabela III: Ver compromissos de um dia e hora

Valor	Tempo
Max	12.891 ms
Min	2.503 ms
Mdia	5.144 ms
Desvio	1.440 ms

(a) Tempo total

Valor	Tempo
Max	12.300 ms
Min	1.819 ms
Mdia	4.366 ms
Desvio	1.445 ms

(b) Tempo de comunição

Tabela IV: Ver compromissos de um dia

Valor	Tempo
Max	10.903 ms
Min	2.121 ms
Mdia	4.523 ms
Desvio	0.191 ms

(a) Tempo total

Valor	Tempo
Max	10.231 ms
Min	1.460 ms
Mdia	3.743 ms
Desvio	0.179 ms

(b) Tempo de comunição

Tabela V: Ver compromissos do mês

4.1 Comparação de tecnologias

O RMI utiliza o protocolo TCP, que uma das suas características é a transferência garantida, assim não foi necessário uma análise de erros na entrega dos pacotes. O que possibilitou uma diminuição do código se comparado a utilização do protocolo UDP em C.

Ao utilizar a tecnologia RMI conseguiu-se uma grande abstração em relação a comunicação entre cliente e servidor, já que a após estabelecida a comunicação o serivodor é chamado através de funções como se não fossem distribuídas. Da mesma maneira os arquivos são vistos como se fossem locais, o que é uma característica de transparência de localização, um dos objetivos de um sistema sistribuído.

5 Conclusão

Utilizar a tecnologia Java RMI facilitou o desenvolvimento de aplicações distribuídas, no qual existe a interação entre um cliente e um servidor, devido a inclusão da implementação do protocolo TCP. Além disso, java proporciona a funcionalidade garbage collector que nos exime de se preocupar com a limpeza de memória, diferentemente do que ocorreu desenvolvendo a agenda na linguagem C.

Por outro lado, existe a necessidade de uma largura de banda consideravelmente maior em relação ao Socket TCP. Entretanto, como a tecnologia Java RMI tem como objetivo fornecer uma transparência de localização e não a eficiência no transporte de dados, o que permitindo um maior nível de abstração e de transparência, auxiliando o programador; a utilização de Java RMI tem uma relação custo-benefício muito boa .

Apesar dos benefícios, escrever código em Java requer um maior conhecimento de orientação a objetos, e o seu desempenho é pífio se comparado à linguagem C.

Referências

[1] Tanenbaum, Andrew S. e Maarten Van Steen Distributed Systems: Principles and Paradigms. Prentice Hall.

[2] Brian "Beej Jorgensen"Hall Beej's Guide to Network Programming - Using Internet Sockets . Disponível em *http://beej.us/guide/bgnet/*, [Último acesso: 07/04/2011].

[3] Tutorial RMI Oracle. Disponível em *http://download.oracle.com/javase/tutorial/rmi/index.html*, [Último acesso: 12/05/2011].

[4] J. Kurose e K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson Addison Wesley, 3 ed., 2005.

6 Anexo

Listing 1: Servidor

```
1  package client;
2  import java.rmi.registry.Registry;
3  import java.rmi.registry.LocateRegistry;
4  import java.rmi.RemoteException;
5  import java.rmi.server.UnicastRemoteObject;
6  import java.io.File;
7  import java.io.FileWriter;
8  import java.io.PrintWriter;
9  import java.io.FileReader;
10 import java.io.RandomAccessFile;
11
12 public class Server implements MC823Server{
13
14     private Opr op = new Opr();
15
16     public Server(){}
17
18     /**
19      * @param args
20      */
21     public static void main(String[] args) {
22
23   try {
24       Server obj = new Server();
25       MC823Server stub = (MC823Server) UnicastRemoteObject.exportObject(obj, 0);
26
27       // Bind the remote object's stub in the registry
28       Registry registry = LocateRegistry.getRegistry();
29       registry.bind("MC823Server", stub);
30
31       System.err.println("Server ready");
32   } catch (Exception e) {
33       System.err.println("Server exception: " + e.toString());
34       e.printStackTrace();
35   }
36
37     }
38
39     public boolean marcarCompromisso(Opr op) throws RemoteException{
40
41   RandomAccessFile f;
42
43   try {
44       //Verifica a existencia da agenda
45       f = new RandomAccessFile(op.getLogin() + ".dat","rw");
46
47       /*Vou ate o final do arquivo*/
48       f.seek(f.length());
49       /*Insere nome compromisso*/
50       f.writeBytes(op.getString());
51       f.writeBytes("\n");
52       /*Insere dia compromisso*/
```

```
53      f.writeBytes(Integer.toString(op.getDia()));
54      f.writeBytes("\n");
55      /*Insere hora compromisso*/
56      f.writeBytes(Integer.toString(op.getHora()));
57      f.writeBytes("\n");
58      /*Insere minuto compromisso*/
59      f.writeBytes(Integer.toString(op.getMinuto()));
60      f.writeBytes("\n");
61
62      f.close();
63
64      return true;
65
66  } catch (Exception e) {
67      System.err.println("File exception: " + e.toString());
68      return false;
69  }
70
71    }
72
73    public boolean IsUsr(Opr op) throws RemoteException{
74  try{
75
76      RandomAccessFile f = new RandomAccessFile("users.dat","r");
77      String usr,psw;
78      while((usr = f.readLine())!= null){
79    psw = f.readLine();
80    /*verifico usuario*/
81    if(usr.equals(op.getLogin())){
82        /*verifico senha do usuario*/
83        System.out.println("Encontrei:\n");
84        System.out.println(op.getPassword());
85        if(psw.equals(op.getPassword()))
86      return true;
87    }
88    System.out.println(usr);
89    System.out.println(psw);
90      }
91
92  } catch (Exception e) {
93      System.err.println("File exception: " + e.toString());
94      return false;
95  }
96
97  return false;
98    }
99
100   public boolean NewUsr(Opr op) throws RemoteException{
101 RandomAccessFile f;
102
103 try{
104
105     f = new RandomAccessFile("users.dat","rw");
106     String usr,psw;
107
108     /*Vou ate o final do arquivo*/
```

```
109       f.seek(f.length());
110       f.writeBytes(op.getLogin());
111       f.writeBytes("\n");
112       f.writeBytes(op.getPassword());
113       f.writeBytes("\n");
114       f.close();
115
116       /*Devo criar agenda para o usuario*/
117       File file = new File(op.getLogin()+".dat");
118       f = new RandomAccessFile(op.getLogin()+".dat","rw");
119       f.close();
120
121   } catch (Exception e) {
122       System.err.println("File exception: " + e.toString());
123       return false;
124   }
125
126   return true;
127     }
128
129     public boolean desmarcarCompromisso(Opr op) throws RemoteException{
130
131   RandomAccessFile f; /*Arquivo*/
132   boolean found = false;
133
134   /*Lista de compromissos nao apagados*/
135   StringBuffer sb = new StringBuffer();
136
137   try {
138       f = new RandomAccessFile(op.getLogin() + ".dat","rw");
139
140       /*Procura compromisso pelo nome*/
141       String name,dia,hora,minuto;
142       while((name = f.readLine())!= null){
143     /*Ignoro dia hora minuto*/
144     dia = f.readLine();
145     hora = f.readLine();
146     minuto = f.readLine();
147
148     System.out.println("\nEstou procurando por:"+op.getString());
149
150     /*verifico se nome procurado nao e o a ser apagado*/
151     if(!(name.equals(op.getString()))){
152         System.out.println("\nNao eh");
153
154         /*Vou manter compromisso na agenda*/
155         sb.append(name+"\n");
156         sb.append(dia+"\n"+hora+"\n"+minuto+"\n");
157
158         /*TO BE DONE*/
159         found = true;
160     }
161       }
162
163       f.close();
164       /*se achei tarefa reescrevo na agenda*/
```

```
165       if(found){
166     op.setString(sb.toString());
167 
168     File trash = new File(op.getLogin() + ".dat");
169     trash.delete();
170     f = new RandomAccessFile(op.getLogin() + ".dat","rw");
171     f.writeBytes(op.getString());
172     f.close();
173 
174       }
175 
176       return found;
177 
178   } catch (Exception e) {
179       System.err.println("File exception: " + e.toString());
180 
181       return found;
182   }
183     }
184 
185     public String obterCompromissoHora(Opr op) throws RemoteException{
186 
187   /*Lista de compromissos*/
188   StringBuffer sb = new StringBuffer();
189   op.setString("Nenhum compromisso nesse dia e horario");
190 
191   try {
192       RandomAccessFile f = new RandomAccessFile(op.getLogin() + ".dat","rw");
193 
194       /*Procura compromisso pelo nome*/
195       String name,dia,hora,minuto;
196       while((name = f.readLine())!= null){
197     /*Ignoro dia hora minuto*/
198     dia = f.readLine();
199     hora = f.readLine();
200     minuto = f.readLine();
201 
202     /*verifico se nome procurado e o mesmo*/
203     if(dia.equals(Integer.toString(op.getDia()))){
204         if(hora.equals(Integer.toString(op.getHora()))){
205 
206       /*Preciso retornar lista de compromissos*/
207       sb.append("-----------\nNome:"+name);
208       sb.append("\nDia: "+dia+"\nHora: "+hora+"\nMinuto: "+minuto + "\n");
209 
210       op.setString(sb.toString());
211 
212         }
213     }
214       }
215       f.close();
216 
217       return op.getString();
218 
219   } catch (Exception e) {
220       System.err.println("File exception: " + e.toString());
```

```
221
222         //retorna a string com o erro
223         return "File exception: " + e.toString();
224     }
225
226       }
227
228       public String obterCompromissoDia(Opr op) throws RemoteException{
229
230     /*Lista de compromissos*/
231     StringBuffer sb = new StringBuffer();
232     op.setString("Nenhum Compromisso nesse dia");
233
234     try {
235         RandomAccessFile f = new RandomAccessFile(op.getLogin() + ".dat","rw");
236
237         /*Procura compromisso pelo nome*/
238         String name,dia,hora,minuto;
239         while((name = f.readLine())!= null){
240       /*Ignoro dia hora minuto*/
241       dia = f.readLine();
242       hora = f.readLine();
243       minuto = f.readLine();
244
245       /*verifico se nome procurado e o mesmo*/
246       if(dia.equals(Integer.toString(op.getDia()))){
247
248           /*Preciso retornar lista de compromissos*/
249           sb.append("-----------\nNome:"+name);
250           sb.append("\nDia: "+dia+"\nHora: "+hora+"\nMinuto: "+minuto + "\n");
251
252           op.setString(sb.toString());
253       }
254         }
255         f.close();
256
257         return op.getString();
258
259     } catch (Exception e) {
260         System.err.println("File exception: " + e.toString());
261
262         //retorna a string com o erro
263         return "File exception: " + e.toString();
264     }
265
266       }
267
268       public String obterCompromissoMes(Opr op) throws RemoteException{
269
270     /*Lista de compromissos*/
271     StringBuffer sb = new StringBuffer();
272     op.setString("Nenhum Compromisso no mes");
273
274     try {
275         RandomAccessFile f = new RandomAccessFile(op.getLogin() + ".dat","rw");
276
```

```
277       /*Procura compromisso pelo nome*/
278       String name,dia,hora,minuto;
279       while((name = f.readLine())!= null){
280     /*Ignoro dia hora minuto*/
281     dia = f.readLine();
282     hora = f.readLine();
283     minuto = f.readLine();
284 
285     /*Preciso retornar lista de compromissos*/
286     sb.append("----------\nNome:"+name);
287     sb.append("\nDia: "+dia+"\nHora: "+hora+"\nMinuto: "+minuto + "\n");
288 
289     op.setString(sb.toString());
290 
291       }
292       f.close();
293 
294       return op.getString();
295 
296   } catch (Exception e) {
297       System.err.println("File exception: " + e.toString());
298 
299       //retorna a string com o erro
300       return "File exception: " + e.toString();
301   }
302 
303     }
304 
305 }
```

Listing 2: Main cliente

```
1  package client;
2  
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.rmi.registry.LocateRegistry;
6  import java.rmi.registry.Registry;
7  
8  public class CMain {
9  
10     private CMain(){}
11 
12     /**
13      * @param args
14      */
15     public static void main(String[] args) {
16 
17   Client user = new Client();
18   BufferedReader leitor = new BufferedReader(new InputStreamReader(System.in));
19   int opSelect;
20   boolean done = false ;
21   MC823Server stub = null;
22 
23   /* Recebo o nome/ip do servidor para a conexao */
24   String host = (args.length < 1) ? null : args[0];
```

```
25  try {
26      /*Procura pelo registro usado pelo host,
27        o registro e usado para referenciar um objeto remoto */
28      Registry registry = LocateRegistry.getRegistry(host);
29      /* Cria o stub para processos distribuidos:
30        toda comunicacao passa por ele.
31        Cria a abstracao da comunicao */
32      stub = (MC823Server) registry.lookup("MC823Server");
33  } catch (Exception e) {
34      System.err.println("Client exception: " + e.toString());
35      e.printStackTrace();
36  }
37
38
39  for(;;){
40      /* Limpando a tela*/
41      //System.out.println((char) 27+ "[2J");
42      /* Inicia com usuario nao cadastrado*/
43      user.NonUserMenu();
44      opSelect = 0;
45      System.out.print("\n Digite a opcao desejada: ");
46
47      for(;;){
48    try {
49        String inBuffer = leitor.readLine();
50        if(inBuffer.length() == 1){
51      opSelect = inBuffer.charAt(0);
52        }
53    } catch (Exception e) {
54    System.err.println("Read exception: " + e.toString());
55    }
56
57    if ((opSelect < '1') || (opSelect > '3'))
58        System.out.println("\n    Opcao invalida, digite a opcao novamente.");
59    else
60        break;
61      }
62      switch(opSelect){
63      case '1':
64    done = user.Login(stub);
65    break;
66      case '2':
67    done = user.NewCal(stub);
68    break;
69      case '3':
70    try{
71        leitor.close();
72    } catch (Exception e){
73        System.err.println("Read exception: " + e.toString());
74    }
75    System.exit(0);
76    done = true;
77      default:
78    done = false;
79      }
80      if(done == true)
```

```
81      break;
82    }
83
84    for(;;){
85        /* Usuario logado, fornecer menu agenda */
86        user.UserMenu();
87
88        for(;;){
89      opSelect = 0;
90      System.out.print("\n    Digite a opcao desejada: ");
91      try {
92          String inBuffer = leitor.readLine();
93          if(inBuffer.length() == 1){
94        opSelect = inBuffer.charAt(0);
95          }
96      } catch (Exception e) {
97          System.err.println("Read exception: " + e.toString());
98      }
99
100     if ((opSelect < '1') || (opSelect > '6'))
101         System.out.println("\n    Opcao invalida, digite a opcao novamente.");
102     else
103         break;
104
105       }
106
107       switch(opSelect){
108       case '1':
109     user.NewComp(stub);
110     break;
111       case '2':
112     user.DelComp(stub);
113     break;
114       case '3':
115     user.ShowHour(stub);
116     break;
117       case '4':
118     user.ShowDay(stub);
119     break;
120       case '5':
121     user.ShowMonth(stub);
122     break;
123       case '6':
124     try{
125         leitor.close();
126     } catch (Exception e){
127         System.err.println("Read exception: " + e.toString());
128     }
129     System.exit(0);
130       default:
131       }
132   }
133
134     }
135 }
```

Listing 3: Cliente

```
1  package client;
2  import java.io.BufferedReader;
3  import java.io.InputStreamReader;
4
5  public class Client {
6
7      private Opr op;
8      private BufferedReader leitor;
9
10     /*_____________________________________________________________
11       |Construtor do usuario -> Representa um objeto usuario      |
12       | Conteudo: uma estrutura para  comunicacao com o servidor |
13       | e string de leitura                                       |
14       |___________________________________________________________|*/
15     public Client() {
16   this.op = new Opr();
17   this.leitor = new BufferedReader(new InputStreamReader(System.in));
18     }
19
20     /*_____________________________________________________________
21       |Operacoes possiveis para o usuario                         |
22       | As funcoes sao:                                           |
23       | Login: Usuario fornece nome se senha que serao comparadas|
24       |        com banco de dados no servidor                     |
25       | UserMenu: Para um usuario ja logado apresenta funcoes de |
26       |           manipulacao na agenda                           |
27       | NonUserMenu: Menu inicial para Usuario nao logado         |
28       | NewComp: Insere novo compromisso na agenda                |
29       | DelComp: Desmarca um compromisso da agenda                |
30       | ShowHour: Mostra compromissos para determinada hora       |
31       | ShowMonth: Mostra compromissos do mes                     |
32       | ShowDay: Mostra compromissos de um dia inteiro            |
33       |___________________________________________________________|*/
34
35
36     /* Funcao: Login do usuario
37        Descricao: compara PassWord e UserName com os do banco de dados
38     */
39     public boolean Login(MC823Server stub) {
40
41   BufferedReader leitor = new BufferedReader(new InputStreamReader(System.in));
42
43   System.out.println("== Procura por usuario ==\n");
44   System.out.print("Digite o nome do usuario:");
45
46   try {
47       this.op.setLogin(leitor.readLine());
48   } catch (Exception e){
49       System.err.println("Read exception: " + e.toString());
50       return false;
51   }
52
53   System.out.print("\nDigite a senha do usuario:");
54   try {
```

```
55        this.op.setPassword(leitor.readLine());
56    } catch (Exception e){
57        System.err.println("Read exception: " + e.toString());
58        return false;
59    }
60
61
62    /*Vou verificar se usuario tem agenda no sistema*/
63    /*E se usuario possui a senha correta*/
64    boolean Ok=false;
65    try{
66        Ok = stub.IsUsr(op);
67        if(Ok == false){
68      System.out.println("\nSenha ou usuarios incorretos\n");
69      return false;
70        }
71    } catch (Exception e) {
72        System.err.println("Client exception: " + e.toString());
73        e.printStackTrace();
74    }
75
76        System.out.println("\nUsuario logado: "+this.op.getLogin());
77        return true;
78      }
79
80      /*Cria agenda de um novo usuario*/
81      public boolean NewCal(MC823Server stub) {
82
83    BufferedReader leitor = new BufferedReader(new InputStreamReader(System.in));
84
85    System.out.println("== Novo usuario ==\n");
86    System.out.print("Digite o nome do novo usuario: ");
87
88    try {
89        this.op.setLogin(leitor.readLine());
90    } catch (Exception e){
91        System.err.println("Read exception: " + e.toString());
92        return false;
93    }
94
95    System.out.print("\nDigite a senha do novo usuario:");
96    try {
97        this.op.setPassword(leitor.readLine());
98    } catch (Exception e){
99        System.err.println("Read exception: " + e.toString());
100       return false;
101   }
102
103   /*Vou criar agenda de usuario no sistema*/
104   boolean Ok=false;
105   try{
106       Ok = stub.NewUsr(op);
107       if(Ok == false){
108     System.out.println("\nNao consegui criar usuarios\n");
109     return false;
110       }
```

```
111   } catch (Exception e) {
112       System.err.println("Client exception: " + e.toString());
113       e.printStackTrace();
114   }
115
116
117   System.out.println("\nUsuario logado: "+this.op.getLogin());
118   System.out.println("\nSenha: "+this.op.getPassword());
119   return true;
120     }
121
122     public void UserMenu() {
123   System.out.println("*********************************************************************************");
124   System.out.println("*   ============            MENU USUARIO           =============        *");
125   System.out.println("*   1. Marcar um compromisso.                                             *");
126   System.out.println("*   2. Desmarcar um compromisso.                                          *");
127   System.out.println("*   3. Ver todos compromissos marcados para um horario de um dia        *");
128   System.out.println("*   4. Ver todos compromissos marcados para um dia.                       *");
129   System.out.println("*   5. Ver todos compromissos do mes.                                     *");
130   System.out.println("*   6. Sair.                                                              *");
131   System.out.println("*                                                                         *");
132   System.out.println("*********************************************************************************");
133     }
134     public void NonUserMenu() {
135   System.out.println("*********************************************************************************");
136   System.out.println("*   ============            MENU INICIAL           =============        *");
137   System.out.println("*   1. Entrar com um usuario.                                             *");
138   System.out.println("*   2. Criar um usuario.                                                  *");
139   System.out.println("*   3. Sair.                                                              *");
140   System.out.println("*                                                                         *");
141   System.out.println("*********************************************************************************");
142     }
143
144     public void NewComp(MC823Server stub) {
145
146   BufferedReader leitor = new BufferedReader(new InputStreamReader(System.in));
147
148   try {
149       System.out.print("\nDigite o nome do compromisso: ");
150       op.setString(leitor.readLine());
151
152       System.out.print("\nDigite o dia do compromisso: ");
153       op.setDia(Integer.parseInt(leitor.readLine()));
154
155       System.out.print("\nDigite o hora do compromisso: ");
156       op.setHora(Integer.parseInt(leitor.readLine()));
157
158       System.out.print("\nDigite os minutos do compromisso: ");
159       op.setMinuto(Integer.parseInt(leitor.readLine()));
160
161
162   } catch (Exception e) {
163       System.err.println("Read exception: " + e.toString());
164   }
165
166   try {
```

```
167             timeStamp temp = new timeStamp();
168       if(stub.marcarCompromisso(op)){
169     System.out.print("\nA operacao foi um sucesso:Compromisso marcado.\nPressione ENTER para continuar...");
170     leitor.readLine();
171       } else {
172     System.out.print("\nErro!!! Server Exception.\nPressione ENTER para continuar...");
173     leitor.readLine();
174       }
175       temp.pararTempo("1/clientTime.dat");
176   } catch (Exception e) {
177       System.err.println("Client exception: " + e.toString());
178       e.printStackTrace();
179   }
180
181     }
182
183     public void DelComp(MC823Server stub) {
184
185   try {
186       System.out.print("\nDigite o nome do compromisso: ");
187       op.setString(leitor.readLine());
188
189   } catch (Exception e) {
190       System.err.println("Read exception: " + e.toString());
191   }
192
193   try {
194         timeStamp temp = new timeStamp();
195       if(stub.desmarcarCompromisso(op)){
196     System.out.print("\nA operacao foi um sucesso: Compromisso desmarcado.\nPressione ENTER para continuar...")
          ;
197     leitor.readLine();
198       } else {
199     System.out.print("\nErro!!! Server Exception.\nPressione ENTER para continuar...");
200     leitor.readLine();
201       }
202       temp.pararTempo("2/clientTime.dat");
203   } catch (Exception e) {
204       System.err.println("Client exception: " + e.toString());
205       e.printStackTrace();
206   }
207
208     }
209
210     public void ShowHour (MC823Server stub) {
211
212   try {
213       System.out.print("\nDigite o dia do compromisso: ");
214       op.setDia(Integer.parseInt(leitor.readLine()));
215
216       System.out.print("\nDigite o horario do compromisso: ");
217       op.setHora(Integer.parseInt(leitor.readLine()));
218
219   } catch (Exception e){
220       System.err.println("Read exception: " + e.toString());
221   }
```

```
222
223 try {
224           timeStamp temp = new timeStamp();
225
226     //Recebe uma string de erro caso aconteca exception on server
227     op.setString(stub.obterCompromissoHora(op));
228      temp.pararTempo("3/clientTime.dat");
229 } catch (Exception e) {
230     System.err.println("Client exception: " + e.toString());
231     e.printStackTrace();
232 }
233
234 try {
235     System.out.println("\n----------\nCompromissos do dia: " + op.getDia()+"e hora"+ op.getHora()+ "h\n
          ----------");
236     System.out.println(op.getString());
237     System.out.println("----------\n");
238     System.out.print("\nPressione ENTER para continuar...");
239     leitor.readLine();
240
241 } catch (Exception e) {
242     System.err.println("Read exception: " + e.toString());
243 }
244
245
246   }
247
248   public void ShowDay(MC823Server stub) {
249 int i;
250
251 try {
252
253     System.out.print("\nDigite o dia do compromisso: ");
254     op.setDia(Integer.parseInt(leitor.readLine()));
255
256 } catch (Exception e) {
257     System.err.println("Read exception: " + e.toString());
258 }
259
260 try {
261     timeStamp temp = new timeStamp();
262
263     //Recebe uma string de erro caso aconteca exception on server
264     op.setString(stub.obterCompromissoDia(op));
265     temp.pararTempo("4/clientTime.dat");
266 } catch (Exception e) {
267     System.err.println("Client exception: " + e.toString());
268     e.printStackTrace();
269 }
270
271 try {
272
273     System.out.println("\n----------\nCompromissos do dia: " + op.getDia()+"\n----------");
274     System.out.println(op.getString());
275     System.out.println("----------\n");
276
```

```
277
278         System.out.print("\nPressione ENTER para continuar...");
279         leitor.readLine();
280
281     } catch (Exception e) {
282         System.err.println("Read exception: " + e.toString());
283     }
284       }
285
286       public void ShowMonth(MC823Server stub) {
287
288     try {
289               timeStamp temp = new timeStamp();
290
291         //Recebe uma string de erro caso aconteca exception on server
292         op.setString(stub.obterCompromissoMes(op));
293            temp.pararTempo("5/clientTime.dat");
294     } catch (Exception e) {
295         System.err.println("Client exception: " + e.toString());
296         e.printStackTrace();
297     }
298
299     try {
300         System.out.println("\n----------\nCompromissos do mes\n----------");
301         System.out.println(op.getString());
302         System.out.println("----------\n");
303
304         System.out.print("\nPressione ENTER para continuar...");
305         leitor.readLine();
306     } catch (Exception e) {
307         System.err.println("Read exception: " + e.toString());
308     }
309       }
310 }
```

Listing 4: Server Interface

```
1  package client;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  public interface MC823Server extends Remote {
7      boolean IsUsr(Opr op) throws RemoteException;
8      boolean NewUsr(Opr op) throws RemoteException;
9      boolean marcarCompromisso(Opr op) throws RemoteException;
10     boolean desmarcarCompromisso(Opr op) throws RemoteException;
11     String obterCompromissoHora(Opr op) throws RemoteException;
12     String obterCompromissoDia(Opr  op) throws RemoteException;
13     String obterCompromissoMes(Opr op) throws RemoteException;
14 }
```

Listing 5: Struct de compromissos

```
1  package client;
2  import java.io.Serializable;
```

```
3
4  /* Armazena as informacoes necessarias na classe Opr */
5  public class Opr implements Serializable{
6    private int operacao;
7    private int dia;
8    private int hora;
9    private int minuto;
10   private String string;
11   private String login;
12   private String password;
13
14   public int getDia() {
15     return dia;
16   }
17   public void setDia(int dia) {
18     this.dia = dia;
19   }
20   public int getHora() {
21     return hora;
22   }
23   public void setHora(int hora) {
24     this.hora = hora;
25   }
26   public int getMinuto() {
27     return minuto;
28   }
29   public void setMinuto(int minuto) {
30     this.minuto = minuto;
31   }
32
33   public String getLogin() {
34     return login;
35   }
36   public void setLogin(String login) {
37     this.login = login;
38   }
39   public String getPassword() {
40     return password;
41   }
42   public void setPassword(String password) {
43     this.password = password;
44   }
45   public int getOperacao() {
46     return operacao;
47   }
48   public void setOperacao(int operacao) {
49     this.operacao = operacao;
50   }
51   public String getString() {
52     return string;
53   }
54   public void setString(String string) {
55     this.string = string;
56   }
57 }
```