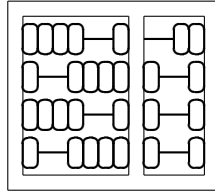


UNIVERSIDADE ESTADUAL DE CAMPINAS

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	2
1.1	Teoria	2
2	Servidor de agenda	2
2.1	Menu inicial	3
2.1.1	Login	3
2.1.2	Novo usuário	3
2.2	Menu usuário	3
2.2.1	Inserção de compromisso	4
2.2.2	Remoção de compromisso	4
2.2.3	Pesquisas	4
3	Ambiente de implementação	4
4	Tempos de comunicação e total	4
4.1	Comparação de tecnologias	6
5	Conclusão	7
6	Anexo	8

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, referencia 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.

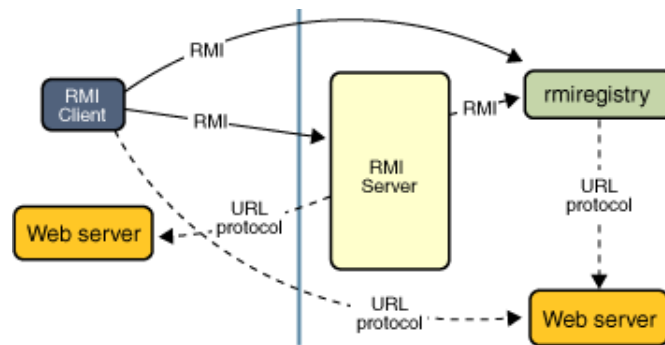


Figura 1: Aplicação distribuída RMI que usa o RMI registry para obter a referência para um objeto remoto.

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 e o tempo de comunicação de cada função. Para o tempo total, pega-se, no cliente, a diferença do tempo no final da função e o tempo quando a função é chamada.

Para o tempo de comunicação, pega-se o tempo total e subtrai-se o tempo de processamento do servidor. O tempo do servidor é calculado fazendo-se a diferença de dois tempos: antes do retorno da função e depois da chamada da função. Para o tempo total das funções obteve-se o tempo de inserir um compromisso, remover o compromisso, ver a agenda do mês, ver a agenda de um dia e 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
Média	3.318 ms
Desvio	0.145 ms

(a) Tempo total

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

(b) Tempo de comunicação

Valor	Tempo
Max	0.517 ms
Min	0.341 ms
Média	0.448 ms
Desvio	0.124 ms

(c) Tempo de processamento

Tabela I: Inserir compromisso

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

(a) Tempo total

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

(b) Tempo de comunicação

Valor	Tempo
Max	12.418 ms
Min	8.687 ms
Média	9.185 ms
Desvio	0.238 ms

(c) Tempo de processamento

Tabela II: Remover compromissos

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

(a) Tempo total

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

(b) Tempo de comunicação

Valor	Tempo
Max	0.836 ms
Min	0.680 ms
Média	0.841 ms
Desvio	0.048 ms

(c) Tempo de processamento

Tabela III: Ver compromissos de um dia e hora

Valor	Tempo	Valor	Tempo	Valor	Tempo
Max	12.891 ms	Max	12.300 ms	Max	0.691 ms
Min	2.503 ms	Min	1.819 ms	Min	0.684 ms
Média	5.144 ms	Média	4.366 ms	Média	0.778 ms
Desvio	1.440 ms	Desvio	1.445 ms	Desvio	0.002 ms

(a) Tempo total (b) Tempo de comunicação (c) Tempo de processamento

Tabela IV: Ver compromissos de um dia

Valor	Tempo	Valor	Tempo	Valor	Tempo
Max	10.903 ms	Max	10.231 ms	Max	0.672 ms
Min	2.121 ms	Min	1.460 ms	Min	0.661 ms
Média	4.523 ms	Média	3.743 ms	Média	0.780 ms
Desvio	0.191 ms	Desvio	0.179 ms	Desvio	0.005 ms

(a) Tempo total (b) Tempo de comunicação (c) Tempo de processamento

Tabela V: Ver compromissos do mês

4.1 Comparação de tecnologias

O RMI utiliza o protocolo TCP, no qual uma das suas características é a transferência garantida de dados, assim não foi necessária uma análise de erros na entrega dos pacotes. O que possibilitou uma diminuição do código em aproximadamente 20% para o protocolo UDP em C e cerca de 38% para o protocolo TCP, também em C.

	TCP	UDP	RMI
Nº de linhas	1344	1038	829

Tabela VI: Comparação do tamanho de código

Ao utilizar a tecnologia RMI conseguiu-se uma grande abstração em relação a comunicação entre cliente e servidor, já que, após estabelecida a comunicação, o servidor 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 distribuído.

Além disso, através do security manager o RMI cuida da segurança do sistema, protegendo o acesso aos recursos do sistema de códigos maliciosos baixados que rodam na máquina virtual Java.

Função	TCP	UDP	RMI
F0	0.705 ms	0.200 ms	2.862 ms
F1	0.725 ms	0.215 ms	2.870 ms
F2	0.705 ms	0.200 ms	2.878 ms
F3	0.715 ms	0.074 ms	3.158 ms
F4	0.727 ms	0.251 ms	4.366 ms
F5	0.716 ms	0.241 ms	3.743 ms

Tabela VII: Comparação de tempos de comunicação

Na tabela acima, nota-se que os tempos de comunicação do RMI é cerca de dez vezes maior que o UDP e para o TCP é, em média, 5 vezes maior. Ao passo que, no desenvolvimento do código o RMI mostrou-se mais fácil de programar que os outros dois protocolos, devido ao mais alto nível de abstração do RMI.

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. Além disso, o RMI cuida da segurança do sistema ao possibilitar a criação e instalação do security manager.

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         if (System.getSecurityManager() == null) {
24             System.setSecurityManager(new SecurityManager());
25         }
26         try {
27             Server obj = new Server();
28             MC823Server stub = (MC823Server) UnicastRemoteObject.exportObject(obj, 0);
29
30             // Bind the remote object's stub in the registry
31             Registry registry = LocateRegistry.getRegistry();
32             registry.bind("MC823Server", stub);
33
34             System.err.println("Server ready");
35         } catch (Exception e) {
36             System.err.println("Server exception: " + e.toString());
37             e.printStackTrace();
38         }
39
40     }
41
42     public boolean marcarCompromisso(Opr op) throws RemoteException{
43
44         RandomAccessFile f;
45
46         try {
47             //Verifica a existencia da agenda
48             f = new RandomAccessFile(op.getLogin() + ".dat", "rw");
49
50             /*Vou ate o final do arquivo*/
51             f.seek(f.length());
```

```

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

```

```

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

```

```

164     }
165
166     f.close();
167     /*se achei tarefa reescrevo na agenda*/
168     if(found){
169         op.setString(sb.toString());
170
171         File trash = new File(op.getLogin() + ".dat");
172         trash.delete();
173         f = new RandomAccessFile(op.getLogin() + ".dat", "rw");
174         f.writeBytes(op.getString());
175         f.close();
176
177     }
178
179     return found;
180
181 } catch (Exception e) {
182     System.err.println("File exception: " + e.toString());
183
184     return found;
185 }
186 }
187
188 public String obterCompromissoHora(Opr op) throws RemoteException{
189
190     /*Lista de compromissos*/
191     StringBuffer sb = new StringBuffer();
192     op.setString("Nenhum compromisso nesse dia e horario");
193
194     try {
195         RandomAccessFile f = new RandomAccessFile(op.getLogin() + ".dat", "rw");
196
197         /*Procura compromisso pelo nome*/
198         String name, dia, hora, minuto;
199         while((name = f.readLine()) != null){
200             /*Ignoro dia hora minuto*/
201             dia = f.readLine();
202             hora = f.readLine();
203             minuto = f.readLine();
204
205             /*verifico se nome procurado e o mesmo*/
206             if(dia.equals(Integer.toString(op.getDia()))){
207                 if(hora.equals(Integer.toString(op.getHora()))){
208
209                     /*Preciso retornar lista de compromissos*/
210                     sb.append("-----\nNome: "+name);
211                     sb.append("\nDia: " +dia+"\nHora: " +hora+"\nMinuto: " +minuto + "\n");
212
213                     op.setString(sb.toString());
214
215                 }
216             }
217         }
218         f.close();
219

```

```

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

```

```

276
277     try {
278         RandomAccessFile f = new RandomAccessFile(op.getLogin() + ".dat", "rw");
279
280         /*Procura compromisso pelo nome*/
281         String name, dia, hora, minuto;
282         while((name = f.readLine()) != null){
283             /*Ignoro dia hora minuto*/
284             dia = f.readLine();
285             hora = f.readLine();
286             minuto = f.readLine();
287
288             /*Preciso retornar lista de compromissos*/
289             sb.append("-----\nNome:" + name);
290             sb.append("\nDia: " + dia + "\nHora: " + hora + "\nMinuto: " + minuto + "\n");
291
292             op.setString(sb.toString());
293
294         }
295         f.close();
296
297         return op.getString();
298     }
299     catch (Exception e) {
300         System.err.println("File exception: " + e.toString());
301
302         //retorna a string com o erro
303         return "File exception: " + e.toString();
304     }
305
306 }
307
308 }

```

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

```

```

77     } catch (Exception e){
78         System.err.println("Read exception: " + e.toString());
79     }
80     System.exit(0);
81     done = true;
82     default:
83     done = false;
84     }
85     if(done == true)
86     break;
87 }
88
89 for(;;){
90     /* Usuario logado, fornecer menu agenda */
91     user.UserMenu();
92
93     for(;;){
94         opSelect = 0;
95         System.out.print("\n    Digite a opcao desejada: ");
96         try {
97             String inBuffer = leitor.readLine();
98             if(inBuffer.length() == 1){
99                 opSelect = inBuffer.charAt(0);
100             }
101         } catch (Exception e) {
102             System.err.println("Read exception: " + e.toString());
103         }
104
105         if ((opSelect < '1') || (opSelect > '6'))
106             System.out.println("\n    Opcao invalida, digite a opcao novamente.");
107         else
108             break;
109
110     }
111
112     switch(opSelect){
113         case '1':
114             user.NewComp(stub);
115             break;
116         case '2':
117             user.DelComp(stub);
118             break;
119         case '3':
120             user.ShowHour(stub);
121             break;
122         case '4':
123             user.ShowDay(stub);
124             break;
125         case '5':
126             user.ShowMonth(stub);
127             break;
128         case '6':
129             try{
130                 leitor.close();
131             } catch (Exception e){
132                 System.err.println("Read exception: " + e.toString());

```



```

133     }
134     System.exit(0);
135     default:
136     }
137 }
138
139 }
140 }

```

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 e 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
135  public void NonUserMenu() {
136      System.out.println("*****");
137      System.out.println("**      =====          MENU INICIAL          =====      **");
138      System.out.println("**      1. Entrar com um usuario.                      **");
139      System.out.println("**      2. Criar um usuario.                          **");
140      System.out.println("**      3. Sair.                                       **");
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             ;
198             leitor.readLine();
199         } else {
200             System.out.print("\nErro!!! Server Exception.\nPressione ENTER para continuar...");
201             leitor.readLine();
202         }
203         temp.pararTempo("2/clientTime.dat");
204     } catch (Exception e) {
205         System.err.println("Client exception: " + e.toString());
206         e.printStackTrace();
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 aconteça 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 }

```

Listing 6: Marcação de tempos

```

1 package client;
2 import java.io.File;
3 import java.io.FileWriter;
4 import java.io.PrintWriter;
5 import java.io.RandomAccessFile;
6
7
8 public class timeStamp {
9
10     RandomAccessFile f;
11     private long start, finish;
12
13     public timeStamp(){
14         try {
15             start = System.nanoTime();
16         } catch (Exception e) {
17             System.err.println("File exception: " + e.toString());
18         }
19     }
20
21     public void pararTempo(String nomeArq){
22         finish = (System.nanoTime() - start) / ((long)1000);
23         try {
24             f = new RandomAccessFile(nomeArq, "rw");
25             f.seek(f.length());
26             f.writeBytes(finish + "\n");
27             f.close();
28         } catch (Exception e) {
29             System.err.println("File exception: " + e.toString());
30         }
31     }
32 }

```