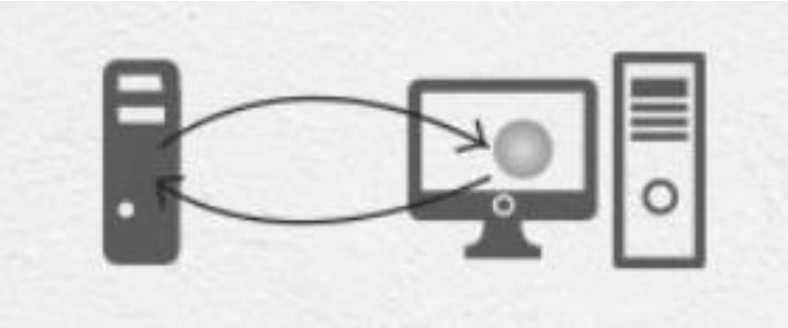
Universidade do Minho
Escola de Engenharia

Aplicações Distribuídas

TP1

Estruturas de Dados, Cliente/Servidor utilizando RMI
Controlo de Concorrência



João Miguel da Silva Alves (83624)

Paulo Jorge Alves (84480)

MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA
INFORMÁTICA MÉDICA 2020/2021

Resumo

Este trabalho prático tem como principal objetivo a familiarização com a linguagem de programação JAVA, nomeadamente o desenvolvimento de estruturas de dados através de uma arquitetura cliente-servidor que implementa RMI e controlo de concorrência.

Para a realização do mesmo procedeu-se ao levantamento de requisitos e à definição das entidades. Depois disto, escreveu-se o código para todos os métodos necessários (registo de pessoas, consultas e exames, realização de consultas, entre outros). Por fim, criou-se um programa (Servidor e Cliente) para o manuseamento dos métodos implementados.

Ao longo do trabalho será explicado todo o trabalho feito, assim como o seu funcionamento.

Índice

1. Introdução

1.1. Contextualização

Um sistema distribuído corresponde a um sistema que possui componentes localizados em computadores independentes entre si e interligados em rede, que se comunicam e coordenam as suas ações através da troca de mensagens entre os componentes (aplicações e serviços).[5]

As principais características de um sistema distribuído são a heterogeneidade, ser um sistema aberto, a escalabilidade (capacidade de funcionar bem quando o número de usuários é elevado), o tratamento de falhas, a transparência e a concorrência de componentes.

Um exemplo de um sistema distribuído é a internet. [3]

Num modelo cliente-servidor o processamento da informação é dividido em diferentes processos. Um processo é responsável pela manutenção da informação (servidores) e outro é responsável pela obtenção dos dados (clientes).

Desta forma, o cliente comunica com o servidor, o servidor recebe a mensagem, interpreta-a e devolve a resposta para o cliente.

O Servidor e o Cliente estão passíveis de comunicar porque usam as mesmas regras ou protocolos, definidos por este modelo. [5][4]

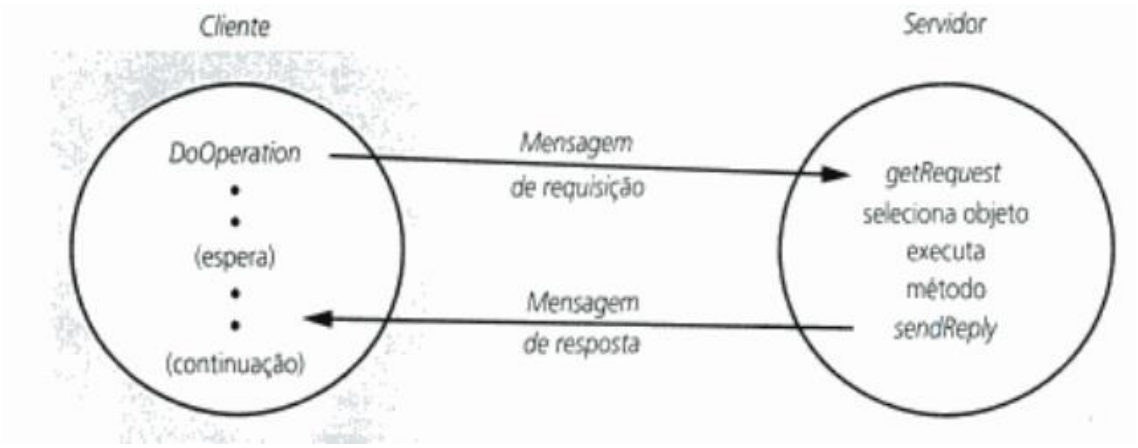


Figura 1 - Modelo Cliente-Servidor

O paralelismo resulta num melhor desempenho dos programas. No entanto podem ocorrer problemas no acesso concorrente a dados e recursos (dados podem-se tornar inconsistentes).

Para tal não acontecer, é necessário fazer um controlo de concorrência("*synchronized*"). Os mecanismos de controlo de concorrência limitam o acesso concorrente a dados e recursos partilhados, garantem o isolamento entre processos e *threads* concorrentes e evitam inconsistências nos dados. [1]

Por fim, o RMI (*Remote Method Invocation*) é uma tecnologia suportada pela plataforma Java que facilita o desenvolvimento de aplicações distribuídas e que permite ao programador invocar métodos de objetos remotos (alojados em máquinas virtuais Java distintas) de uma forma muito semelhante à invocação de objetos locais.

Uma noção central ao RMI é o da separação entre interface e implementação de uma classe. A particularidade mais relevante desta separação é que o RMI permite que a interface e a respetiva implementação se localizem em *JVM's* diferentes. O RMI torna possível que uma determinada aplicação cliente adquira uma interface referente a uma classe que corre numa *JVM* diferente.

A comunicação entre a interface e a implementação é assegurada pelo RMI recorrendo a TCP/IP. A próxima figura tenta ilustrar este processo em alto nível. [5][2]

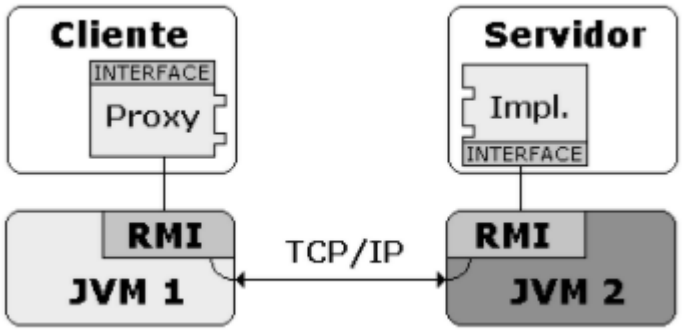


Figura 2 - RMI: Comunicação entre a interface e a implementação

1.2. Apresentação do Caso de Estudo

Para este trabalho foi proposto a implementação de um sistema multiutilizador de gestão de uma clínica (registo de dados em ficha clínica, agenda de consultas, …).

Para desenvolver o projeto usou-se o IntelliJ IDEA, um ambiente de desenvolvimento integrado escrito em Java para o desenvolvimento de software de computador.

1.3. Objetivos

O principal objetivo deste trabalho é a familiarização da linguagem de programação JAVA, nomeadamente: Estruturas de dados, cliente servidor usando RMI e controlo de concorrência.

1.4. Estrutura do Relatório

Este relatório divide-se fundamentalmente em 3 partes.

Inicialmente, é introduzida uma noção geral sobre o tema. Segue-se o levantamento dos requisitos e a definição das entidades envolvidas. Por fim, é feita uma breve explicação do código usado em todo o processo (classes, métodos, interfaces, carregamento de dados, servidor e cliente).

2. Levantamento de Requisitos

Sendo o objetivo deste trabalho a implementação de um sistema multiutilizador de gestão de uma clínica, achou-se pertinente a utilização de dados relativos a utentes, a médicos, a funcionários e aos gestores da instituição de saúde.

O gestor é o responsável pelo registo de novos médicos e funcionários na instituição. Para adicionar novos médicos, este tem de introduzir o nome do médico, a morada, NIF, número do CC, data de nascimento, número da cédula e a respetiva especialidade. Para adicionar novos funcionários, o gestor tem de introduzir o nome do funcionário, a morada, NIF, número do CC, data de nascimento e o número de funcionário correspondente.

O funcionário pode registar um utente na IS, introduzindo o nome do utente, morada, NIF, número do CC, data de nascimento, número de utente, número de telefone e email. Desta forma é criada automaticamente uma ficha médica deste novo paciente. Uma vez registado e feito o login, o utente pode agendar uma consulta de determinada especialidade, a uma certa hora. A consulta será agendada se existirem médicos da especialidade para a realizar na hora pretendida, caso contrário não consegue agendar. Esta consulta, passível de ser cancelada até à data da sua realização, fica registada na agenda até ao momento da consulta. Neste momento, o médico inicia a consulta, introduzindo as medições efetuadas durante a consulta e as respetivas observações e prescrições. Caso seja necessário a realização de exames médicos, o médico podê-los-á agendar para uma data posterior à consulta.

Por fim, o gestor tem acesso a outras funcionalidades, como por exemplo ver a lista de utentes, funcionários e médicos da IS. O médico, além do que já foi descrito anteriormente, pode por exemplo, ver a ficha médica de um determinado utente ou consultar a sua lista de consultas. O utente pode consultar as consultas que realizou, tal como os respetivos exames. Isto são apenas alguns exemplos de funcionalidades que os utilizadores do sistema podem ter acesso.

3. Entidades

3.1. Entidade “Pessoa”

Os atributos necessários para caracterizar a entidade Pessoa são: nome, morada, nif (número de identificação fiscal), cc (número do CC) e datanasc (data de nascimento).

Analisando agora as entidades Utente, Médico, Funcionário e Gestor, vemos que todas podem (e devem) ter os atributos da entidade Pessoa; portanto, criaram-se subclasses de Pessoa para representá-los.

3.2. Entidade “Utente”

Como referido anteriormente, a entidade Utente é uma subclasse da entidade Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos desta entidade.

Além destes, os atributos necessários para caracterizar a entidade Utente são: numutente (número de utente), telefone, telefone de emergência e email.

3.3. Entidade “Médico”

Como referido anteriormente, a entidade Médico é uma subclasse da entidade Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos desta entidade.

Além destes, os atributos necessários para caracterizar a entidade Médico são: cédula e especialidade.

3.4. Entidade “Funcionário”

Como referido anteriormente, a entidade Funcionário é uma subclasse da entidade Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos desta entidade.

Além destes, o único atributo necessário para caracterizar a entidade Funcionário é o numfunc (número de funcionário).

3.5. Entidade “Gestor”

Como referido anteriormente, a entidade Gestor é uma subclasse da entidade Pessoa, ou seja, os atributos da Pessoa continuam a ser atributos desta entidade.

Além destes, os atributos necessários para caracterizar a entidade Gestor são: telefone, telefone de emergência e email.

3.6. Entidade “Consulta”

Esta entidade refere-se às consultas realizadas. Os atributos desta entidade são: data (data da realização da consulta), observações, uma lista de prescrições, uma lista de Exames (que são marcados na consulta e posteriormente atualizados) e o atributo médico (médico que realiza a consulta).

3.7. Entidade “Medicamento”

Esta entidade refere-se aos medicamentos. Os atributos desta entidade são: dci (denominação comum internacional), nome, forma farmacêutica, dosagem, estado de autorização, genérico e titular_aim.

3.8. Entidade “Prescrição”

Esta entidade refere-se às prescrições que o médico prescreve na consulta. Os atributos desta entidade são: data, medicamento e toma.

3.9. Entidade “Medição”

Esta entidade refere-se às medições realizadas em cada consulta. Os atributos desta entidade são: data, peso, altura, glicemia, tensão arterial, colesterol, triglicerídeos, saturação e, por último, o inr (*international normalized ratio*).

3.10. Entidade “Exame”

Esta entidade refere-se aos exames realizados, que são marcados na consulta. Os atributos desta entidade são: id exame, id utente, tipo, local, data, hora, duração do exame, preço, estado e uma lista de observações.

3.11. Entidade “FichaMedica”

Esta entidade refere-se às fichas médicas de cada utente. Assim, um dos atributos é o Utente. Além deste, os restantes atributos são: agregado, medições (realizadas durante a consulta), prescrições crónicas, histórico e o atributo consultas (contém todas as consultas realizadas pelo utente).

3.12. Entidade” EntradaAgenda”

Esta entidade refere-se ao local onde o utente pode marcar as consultas que pretende efetuar. Assim, um dos atributos é o Utente. Além deste, os restantes atributos são: data, hora, duração, médico e estado (quando o utente marca uma consulta, esta fica como “marcada” até à hora da sua realização).

3.13. Entidade “Agenda”

Esta entidade corresponde a um conjunto de entradas de agenda. Este é o único atributo existente nesta entidade.

4. Implementação do projeto

4.1. Classes

O modelo implementado baseia-se num servidor central que contém uma base de dados, a caracterização e a manipulação dos objetos compartilhados.

As classes implementadas agrupam objetos com uma semelhante estrutura de dados, onde os métodos e propriedades foram definidos com base no raciocínio usada construção das entidades mencionadas na descrição do projeto.

Cada classe possui os seus respetivos atributos, construtores e os métodos get-set.

Nos 3 exemplos abaixo, a classe pessoa é estendida à classe utente que, por sua vez, possui atributos que se encontram nessa classe, nomeadamente “nome”, “morada”, “nif” e “cc”. Além disso, a classe utente possui também atributos como “numutente”, “telefone”, “telefone_emergencia”, “email”. A classe “EntradaAgenda” representa a o momento de agenda de uma consulta, ou seja, possui atributos como “data”, “hora”, “duração”, “medico”, “utente” e “estado” que terá o valor de marcado quando a consulta for marcada.

4.1.1. Classe Pessoa

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public abstract class Pessoa implements Serializable {
7.
8.      private String nome;
9.      private String morada;
10.     private String nif;
11.     private String cc;
12.     private LocalDate datanasc;
13.
14.     public Pessoa() {
15.     }
16.
17.     public Pessoa(String nome, String morada, String nif, String cc, LocalDate datanasc)
    {
18.         this.nome = nome;
19.         this.morada = morada;
20.         this.nif = nif;
21.         this.cc = cc;
22.         this.datanasc = datanasc;
23.     }
24.
25.     public String getNome() {
26.         return nome;
27.     }
```

```
28.
29.     public void setNome(String nome) {
30.         this.nome = nome;
31.     }
32.
33.     public String getMorada() {
34.         return morada;
35.     }
36.
37.     public void setMorada(String morada) {
38.         this.morada = morada;
39.     }
40.
41.     public String getNif() {
42.         return nif;
43.     }
44.
45.     public void setNif(String nif) {
46.         this.nif = nif;
47.     }
48.
49.     public String getCc() {
50.         return cc;
51.     }
52.
53.     public void setCc(String cc) {
54.         this.cc = cc;
55.     }
56.
57.     public LocalDate getDatanasc() {
58.         return datanasc;
59.     }
60.
61.     public void setDatanasc(LocalDate datanasc) {
62.         this.datanasc = datanasc;
63.     }
64.
65.     @Override
66.     public String toString() {
67.         return "Pessoa{" +
68.                 "nome='" + nome + '\'' +
69.                 ", morada='" + morada + '\'' +
70.                 ", nif='" + nif + '\'' +
71.                 ", cc='" + cc + '\'' +
72.                 ", datanasc=" + datanasc +
73.                 '}';
74.     }
75. }
```

4.1.2. Classe Utente

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Utente extends Pessoa  implements Serializable {
7.
8.    private String numutente;
9.    private String telefone;
10.   private String telefone_emergencia;
11.   private String email;
```

```
12.
13.     public Utente(String numutente, String telefone, String telefone_emergencia, String email) {
14.         this.numutente = numutente;
15.         this.telefone = telefone;
16.         this.telefone_emergencia = telefone_emergencia;
17.         this.email = email;
18.     }
19.
20.     public Utente(String nome, String morada,
21.                   String nif,
22.                   String cc, LocalDate datanasc,
23.                   String numutente, String telefone,
24.                   String telefone_emergencia, String email) {
25.         super(nome, morada, nif, cc, datanasc);
26.         this.numutente = numutente;
27.         this.telefone = telefone;
28.         this.telefone_emergencia = telefone_emergencia;
29.         this.email = email;
30.     }
31.
32.     public String getNumutente() {
33.         return numutente;
34.     }
35.
36.     public void setNumutente(String numutente) {
37.         this.numutente = numutente;
38.     }
39.
40.     public String getTelefone() {
41.         return telefone;
42.     }
43.
44.     public void setTelefone(String telefone) {
45.         this.telefone = telefone;
46.     }
47.
48.     public String getTelefone_emergencia() {
49.         return telefone_emergencia;
50.     }
51.
52.     public void setTelefone_emergencia(String telefone_emergencia) {
53.         this.telefone_emergencia = telefone_emergencia;
54.     }
55.
56.     public String getEmail() {
57.         return email;
58.     }
59.
60.     public void setEmail(String email) {
61.         this.email = email;
62.     }
63.
64.     @Override
65.     public String toString() {
66.         return "Utente{" +
67.                 super.toString() +
68.                 "numutente='" + numutente + '\'' +
69.                 ", telefone='" + telefone + '\'' +
70.                 ", telefone_emergencia='" + telefone_emergencia + '\'' +
71.                 ", email='" + email + '\'' +
72.                 '}';
73.     }
74. }
```

4.1.3. Classe EntradaAgenda

```
1.  package aula2.processoclinico;
2.  import java.io.Serializable;
3.  import java.time.LocalDate;
4.  import java.time.LocalDateTime;
5.  import java.time.LocalTime;
6.  import java.util.List;
7.  import java.util.Map;
8.  import java.util.ArrayList;
9.
10. public class EntradaAgenda implements Serializable {
11.
12.     private LocalTime hora;
13.     private LocalDate data;
14.     private LocalTime duracao = LocalTime.parse("00:30:00");
15.     private Medico medico;
16.     private Utente utente;
17.     private String estado;
18.
19.     public EntradaAgenda(LocalTime hora, LocalDate data, Medico medico, Utente utente, S
    tring estado) {
20.         this.hora = hora;
21.         this.data = data;
22.         this.medico = medico;
23.         this.utente = utente;
24.         this.estado = estado;
25.     }
26.
27.     public LocalTime getHora() {
28.         return hora;
29.     }
30.
31.     public void setHora(LocalTime hora) {
32.         this.hora = hora;
33.     }
34.
35.     public LocalDate getData() {
36.         return data;
37.     }
38.
39.     public void setData(LocalDate data) {
40.         this.data = data;
41.     }
42.
43.     public LocalTime getDuracao() {
44.         return duracao;
45.     }
46.
47.
48.     public Medico getMedico() {
49.         return medico;
50.     }
51.
52.     public void setMedico(Medico medico) {
53.         this.medico = medico;
54.     }
55.
56.     public Utente getUtente() {
57.         return utente;
58.     }
59.
60.     public void setUtente(Utente utente) {
61.         this.utente = utente;
62.     }
```

```
63.     public String getEstado() {
64.         return estado;
65.     }
66.
67.     public void setEstado(String estado) {
68.         this.estado = estado;
69.     }
70.
71.     @Override
72.     public String toString() {
73.         return "EntradaAgenda{" +
74.                 "hora=" + hora +
75.                 ", data=" + data +
76.                 ", duracao=" + duracao +
77.                 ", medico=" + medico +
78.                 ", utente=" + utente +
79.                 ", estado='" + estado + '\'' +
80.                 '}';
81.     }
82. }
```

4.2. PC Manager

Nesta classe implementaram-se todos os métodos que serão necessários para desenvolver a nossa aplicação.

Os dois métodos abaixo recebem como parâmetros os atributos que definem um utente e irá verificar se nas "fichasUtente", que correspondem a um hashmap que contém como chaves os números dos utentes e os valores são os respetivos utentes, esse utente existe ou não. Caso o utente não exista, então será adicionada às "fichasUtente" a chave correspondente ao número do utente e o valor correspondente ao utente respetivo.

De forma análoga é também feita a adição de um funcionário, médico e medicamento. Para tal usam-se "funcionários" que corresponde a um hashmap que contém como chave o número do funcionário e valores os respetivos funcionários, "médicos" que corresponde a um hashmap que contém como chave a cédula do médico e valores os respetivos médicos, e "medicamentos" que corresponde a um hashmap que contém como chave um contador e valores os respetivos medicamentos.

```
1.  @Override
2.  public void AdicionaUtente(String Nome, String morada, String nif, String cc,
3.                                 LocalDate dn, String nutente, String telefone,
4.                                 String telefone_emergencia, String email)
                                   throws RemoteException{
5.          if (fichasUtente.containsKey(nutente)){
6.              System.out.println("Utente já existe");
7.          } else {
8.              Utente u = new Utente(Nome, morada, nif, cc, dn, nutente, telefone,
9.                      telefone_emergencia, email);
```

```
            AdicionarUtente(u);
        }
}
```

```
public void AdicionarUtente(Utente t){
        this.fichasUtente.put(t.getNumutente(), new FichaMedica(t));
}
```

O método “LoginGestor” recebe como parâmetros um número e uma passw. Se o número for igual a 00200 que corresponde ao número do gestor, uma vez que só existe um na nossa clínica e se a passw for igual a “alves.lda”, então é assegurado o login do gestor.

Os logins do utente, funcionário e médico são feitos de forma análoga, com a exceção dos parâmetros que cada método recebe. Se as credenciais forem as corretas então será permitido o login, caso contrário aparecerão avisos a dizer que é necessário registar-se ou que a password está incorreta.

```
@Override
    public synchronized boolean LoginGestor(int numero, String passw )
                                                    throws RemoteException{
        int numeroofic = 00200;
        String password = "alves.lda";
        boolean login = false;
        if (numero == numeroofic && passw.equals(password)){
            login = true;
            return true;
        }
        else{
            return false;
        }
    }
```

O método “verlistaUtentes” mostra o nome de todos os utentes e adiciona aos hasmap “utente_name”, “utente_nif”, “utente_nutente” as chaves de nome, nif e número de utente respetivamente e como valores o respetivo utente.

De forma análoga foram feitos os métodos “verlistaMedicos” e “verlistaFuncionarios”.

```
@Override
   public void verlistaUtentes() throws RemoteException {
       for (FichaMedica fichamedica : fichasUtente.values()) {
           System.out.println(fichamedica.getUtente().getNome());
           utente_name.put(fichamedica.getUtente().getNome(), fichamedica.getUtente());
           utente_nif.put(fichamedica.getUtente().getNif(), fichamedica.getUtente());
           utente_nutente.put(fichamedica.getUtente().getNumutente(), fichamedica.getUte
nte());
       }
 }
```

O método “marca_consulta” recebe como parâmetros um número de um utente, uma data, uma hora e uma especialidade. Se o método “VerificarDisponibilidade” retornar um médico, então a consulta será marcada e será criada uma instância da entrada da agenda e adicionada ao hashset “agenda” que contém várias “entradaagenda” que por sua vez correspondem às consultas agendadas.

O método “Verificar_Disponiblidade” recebe como parâmetros uma data, uma hora e uma especialidade e irá ver para os médicos dessa especialidade as suas consultas agendadas. Se a agenda não possuir consultas agendadas, então irá retornar o primeiro médico com essa especialidade. Caso contrário, irá ver, para cada consulta agendada na agenda, se o médico se encontra nessa consulta. Se se encontrar irá verificar se a data e a hora são iguais aos parâmetros. Se assim acontecer, irá iterar para uma seguinte consulta. Senão, irá averiguar se a hora passada como argumento é superior à hora da consulta mais a duração da consulta. Se for inferior, iterará para uma nova consulta na agenda, caso contrário retornará esse médico. Na eventualidade de todos os médicos estarem ocupados irá ser retornado o único médico dessa especialidade que não se encontra na agenda. Caso não haja nenhum médico dessa especialidade que respeite estas condições então retorna nulo.

```
1.  @Override
2.    public synchronized String marca_consulta(String numutente,LocalDate data, LocalTime
    hora, String especialidade)
3.            throws RemoteException, IOException {
4.        Medico medico = VerificarDisponiblidade(data, hora, especialidade);
5.        if (medico != null) {
6.            String estado = "Marcado";
7.            Utente utente = fornecer_utente(numutente);
8.            EntradaAgenda eag = new EntradaAgenda(hora, data, medico, utente, estado);
9.            agenda.add(eag);
10.       } else {
11.           return "Não é possivel marcar a consulta!";
12.       }
13.       return "Consulta marcada!";
14.   }
```

```
1.  public synchronized Medico VerificarDisponiblidade(LocalDate data, LocalTime hora, Strin
    g especialidade) throws RemoteException{
2.
3.          for (Medico umedico: medicos.values()){
4.              if (umedico.getEspecialidade().equals(especialidade)){
5.                  if (agenda.size() == 0){
6.                      return umedico;
7.                  }
8.                  else {
9.                      int contador=0;
10.                     for (EntradaAgenda eag : agenda) {
11.                         if (!umedico.equals(eag.getMedico())) {
12.                             contador += 1;
13.                         } else {
14.
```

```
15.                             if (data.equals(eag.getData()) && hora.equals(eag.getHora())){
16.                               break;
17.                             } else {
18.                                 LocalTime t1 = eag.getHora().plusHours(eag.getDuracao().
    getHour());
19.                                 LocalTime t2 = t1.plusMinutes(eag.getDuracao().getMinute());

20.                                     if (hora.isBefore(t2)) {
21.                                         break;
22.                                     } else {
23.                                         return umedico;
24.                                     }
25.                                 }
26.                             }
27.                         }
28.                         if (contador == agenda.size()){
29.                             return umedico;
30.                         }
31.
32.                     }
33.                 }
34.             }
35.             return null;
36.     }
```

O método “realizarConsulta” recebe como parâmetros: um número de utente, uma data, uma hora e os atributos para definir uma instância da classe “medição”. Depois, irá percorrer cada “entradaAgenda” se a data e a hora da realização da consulta correspondem à data e à hora dessa “entradaAgenda” e quando isso acontecer irá buscar a ficha médica do utente com esse número passado como argumento e depois irá verificar se no hashmap “consultas” que possui como chaves identificadoras para os valores que correspondem às consultas já realizadas. Se não houver nenhuma consulta realizada, então essa consulta será adicionada à ficha médica do utente, irá ser criada uma instância da classe medição, pois um médico numa consulta faz uma medição e depois irá ser adicionada à ficha médica do utente e também se cria uma entrada no hashmap “consultas”. Caso o hashmap “consultas” esteja preenchido com consultas, então irá verificar para cada uma das consultas se a data e a hora correspondem à data e hora passada como parâmetros. Caso seja, então diz que a consulta já foi realizada, caso contrário faz exatamente o mesmo que na primeira situação mencionada.

```
1.  @Override
2.      public synchronized UUID realizarConsulta(String numu,LocalDate data,LocalTime hora,
     double peso, int altura, double glicemia, int tensaoarterial, int colesterol, int
    triglicerideos, int saturacao, int inr) throws RemoteException {
3.
4.          UUID idco = UUID.randomUUID();
5.          String observacoes = "";
6.          List<Prescricao> prescricoes = new ArrayList<>();
7.          List<Exame> exames = new ArrayList<>();
8.          for (EntradaAgenda eag : agenda) {
9.              if (hora.equals(eag.getHora()) && data.equals(eag.getData()) &&
    eag.getUtente().getNumutente().equals(numu)) {
```

```
                FichaMedica fu = fichasUtente.get(numu);
                Consulta c=new Consulta(eag.getData(),observacoes,prescricoes,exames);
                if (consultas.size() == 0){
                    c.setMedico(eag.getMedico());
                    fu.addConsulta(c);
                    Medicao medicao = adicionar_medicao(data, peso, altura, glicemia, tensaoarterial, colesterol, triglicerideos, saturacao, inr);
                    fu.addMedicao(medicao);
                    consultas.put(idco, c);
                    return idco;
                }
                else {
                    for (UUID key : consultas.keySet()) {
                        if (consultas.get(key).getData() == c.getData()) {
                            System.out.println("Consulta já realizada!");
                            System.out.println(key);
                            return key;
                        } else {
                            c.setMedico(eag.getMedico());
                            fu.addConsulta(c);
                            Medicao medicao = adicionar_medicao(data, peso, altura, glicemia, tensaoarterial, colesterol, triglicerideos, saturacao, inr);
                            fu.addMedicao(medicao);
                            consultas.put(idco, c);
                            return idco;
                        }
                    }
                }
            }
        }
```

O método “adicionarObservacoes” permite adicionar observações a uma determinada consulta. Para tal recebe como parâmetros um id de uma consulta e observações. No fim essas observações serão adicionadas à consulta com esse id.

```
@Override
public synchronized void adicionarObservacoes(UUID id_co,String observacoes) throws RemoteException{

        for (UUID id_consulta : consultas.keySet()) {

            if (id_consulta.equals(id_co)) {
                Consulta co = consultas.get(id_consulta);
                co.addObservacao(observacoes);
                System.out.println("Observacoes adicionadas!");
            }
        }
    }
```

O método “adicionar_prescrições” recebe como parâmetros um id de uma consulta, uma data e atributos que permitem criar uma instância da classe medicamento. De seguida, irá adicionar à consulta com o id passado como parâmetro essa prescrição.

```
1.  @Override
2.  public synchronized void adicionar_prescricoes(UUID id_co,LocalDate data, String toma,
    String dci, String nome, String formafarmaceutica, String dosagem, String
    estadoautorizacao, Boolean generico, String titular_aim) throws RemoteException{
3.          for (UUID id_consulta : consultas.keySet()) {
4.              if (id_consulta.equals(id_co)) {
5.                  Consulta co = consultas.get(id_consulta);
6.                Medicamento medc = new Medicamento(dci, nome, formafarmaceutica, dosagem,
    estadoautorizacao, generico, titular_aim);
7.
8.                  Prescricao presc = new Prescricao(data, medc, toma);
9.                  co.addPrescricao(presc);
10.                 System.out.println("Prescricao adicionada!");
11.
12.             }
13.         }
14.     }
```

O método “adicionar_histórico” recebe como parâmetros um histórico e um número de um utente. De seguida, irá procurar a ficha médica do utente com aquele número e irá adicionar esse histórico a essa ficha médica.

```
1.  @Override
2.      public synchronized void adicionar_historico(String historico, String numutente)
    throws RemoteException{
3.          for(String numero : fichasUtente.keySet()){
4.              if (numero .equals (numutente)){
5.                  FichaMedica fu = fichasUtente.get(numero);
6.                  fu.setHistorico(historico);
7.              }
8.          }
9.      }
```

O método “marcar_exame” recebe como parâmetros um id de uma consulta e os atributos que permitem criar uma instância da classe exame. De seguida, irá encontrar a consulta para o id passado como parâmetro e irá adicionar a essa consulta, o exame que pretende marcar.

```
1.  @Override
2.      public synchronized void marca_exame(UUID id_co,int idu, String tipo, String local,
    LocalDate data, LocalTime hora, LocalTime duracao_exame,double preco, boolean estado,
    ArrayList<String> observacoes) throws RemoteException {
3.
4.          Exame ex;
5.
6.          for (UUID id_consulta: consultas.keySet()){
7.              if (id_consulta.equals(id_co)){
8.                  Consulta co = consultas.get(id_consulta);
9.                  ex = new Exame(idu, tipo, local, data, hora, duracao_exame, preco, estad
    o, observacoes);
10.                 co.addExame(ex);
11.             }
12.         }
13.     }
```

Com o método “cancelar_consulta” é possível cancelar uma consulta. Para tal, irá para todas as consultas que se encontram agendadas verificar qual é que possui o utente, a data, a hora e a especialidade que são passados como parâmetros.

```
1.  @Override
2.      public synchronized int cancelar_consulta(String numutente, LocalDate data,
    LocalTime hora, String especialidade) throws RemoteException{
3.          for(EntradaAgenda eag : agenda ){
4.
5.              if(numutente.equals(eag.getUtente().getNumutente()) && (data.equals(eag.getD
    ata())) && (hora.equals(eag.getHora())) && (especialidade.equals(eag.getMedico().getEspe
    cialidade()))){
6.                  agenda.remove(eag);
7.                  System.out.println("Consulta cancelada!");
8.                  return 1;
9.              }
10.         }
11.         return 0;
12.     }
```

Este método abaixo permite ver qual a ficha médica de um utente que possui o número passado como parâmetro.

```
1.  @Override
2.      public synchronized void ver_fichaMedica(String numutente) throws RemoteException{
3.          for(String numero : fichasUtente.keySet()) {
4.              if (numero .equals (numutente)) {
5.                  FichaMedica fu = fichasUtente.get(numero);
6.                  System.out.println(fu);
7.
8.              }
9.          }
10.     }
```

O método “exames_utente” recebe um número de um utente como parâmetro e mostra a partir da ficha médica e de cada consulta desse utente, todos os exames marcados pelo utente.

```
1.  @Override
2.      public synchronized void exames_utente(String numutente) throws RemoteException{
3.
4.          for(String numero : fichasUtente.keySet()){
5.              if (numero .equals (numutente)){
6.                  FichaMedica fu = fichasUtente.get(numero);
7.
8.                  for(Consulta co : fu.getConsultas().values()){
9.                      System.out.println(co.getExames());
10.                 }
11.             }
12.         }
13.     }
```

O método “consultas_utente” recebe como parâmetro um número de um utente e para a ficha médica desse utente, mostra todas as consultas desse utente.

```
@Override
    public synchronized void consultas_utente(String numutente) throws RemoteException{

       for(String numero : fichasUtente.keySet()){
            if (numero .equals (numutente)){
                FichaMedica fu = fichasUtente.get(numero);
                System.out.println(fu.getConsultas());
            }
        }
    }
```

O método “consultas_médico” recebe um número de um médico que corresponde à sua cédula e irá verificar para a consulta que se encontra agendada se o médico é igual àquele que possui o número passado como parâmetro. E de seguida, mostra essa consulta se assim acontecer.

```
@Override
    public synchronized void consultas_medico(String nummedico) throws RemoteException{

        for(Consulta co : consultas.values()){
            if (co.getMedico().getCedula().equals(nummedico)){
                System.out.println(co);
            }
        }
    }
```

Este último método recebe como parâmetro um ano e retorna todos os utentes que nasceram nesse mesmo ano.

```
@Override
    public synchronized void utentes_idade(int ano) throws RemoteException{
        for (FichaMedica fic: fichasUtente.values()){
            Utente u = fic.getUtente();
            if (u.getDatanasc().getYear() ==  ano){
                System.out.println(u.getNome());
            }
        }
    }
```

Todo o código que foi referido e não documentado, será apresentado em anexo.

4.3. Interfaces

Para a implementação da lógica de negócio é necessário a existência de interfaces. Pode-se definir interface como um plano que faz a comunicação entre dois meios diferentes. É um recurso muito utilizado em Java com o objetivo de forçar as classes a ter métodos ou propriedades em comum para que existiam num determinado contexto.

Dentro das interfaces existem apenas assinaturas de métodos e propriedades, só o "molde", onde a classe tem a função de realizar a sua implementação, dando comportamentos práticos aos métodos.[5]

Assim, para a implementação da lógica de negócio deste projeto foram fornecidas, as interfaces abaixo.

4.3.1. Interface Utente

A interface Utente evidencia o que um utente como cliente poderá realizar. Portanto, o utente poderá efetuar o seu login, poderá marcar uma consulta, verificar quais são as suas consultas que realizou tais como os exames que marcou e ainda a possibilidade de cancelar uma consulta que agendou para uma determinada data, hora e uma especialidade.

```
1.  package aula2.interfaces;
2.  import java.io.IOException;
3.  import java.rmi.Remote;
4.  import java.rmi.RemoteException;
5.  import java.time.LocalDate;
6.  import java.time.LocalDateTime;
7.  import java.time.LocalTime;
8.  import java.util.UUID;
9.
10.
11. public interface UtenteInt extends Remote{
12.
13.  public String LoginUtente(String numutente, String nif ) throws RemoteException;
14.
15.  Public String marca_consulta(String numutente, LocalDate data, LocalTime hora,
    String especialidade) throws RemoteException, IOException;
16.
17.  public void consultas_utente(String numutente) throws RemoteException;
18.
19.  public void exames_utente(String numutente) throws RemoteException;
20.
21.  public int cancelar_consulta(String numutente, LocalDate data, LocalTime hora,
    String especialidade) throws RemoteException;
```

4.3.2. Interface Funcionário

A interface Funcionário mostra o que um funcionário pode fazer na clínica. Este pode efetuar o seu login, pode adicionar um utente à base de dados da clínica, pode buscar um determinado utente e pode guardar as alterações efetuadas.

```
1.  package aula2.interfaces;
2.
3.  import java.io.IOException;
4.  import java.rmi.Remote;
5.  import java.rmi.RemoteException;
6.  import java.rmi.server.UnicastRemoteObject;
7.  import java.time.LocalDate;
8.  import java.time.LocalDateTime;
9.  import java.time.LocalTime;
10. import java.util.ArrayList;
11. import java.util.List;
12. import java.util.Map;
13. import java.util.UUID;
14.
15. import aula2.exceptions.DoesNotExistsException;
16. import aula2.processoclinico.Utente;
17. import aula2.progs.*;
18. import aula2.processoclinico.*;
19.
20.
21. public interface FuncionarioInt extends Remote{
22.
23.  public int LoginFunc (int numfunc, String nif) throws RemoteException;
24.
25.  public void AdicionaUtente(String Nome, String morada, String nif, String cc,
26.                       LocalDate dn, String nutente, String telefone,
27.                       String telefone_emergencia, String email) throws RemoteException;
28.  public Utente getUtente(String nome) throws DoesNotExistsException,RemoteException;
29.  public Utente getUtente(String nif) throws DoesNotExistsException,RemoteException;
30.  public Utente getUtente(String numutente) throws DoesNotExistsException,
    RemoteException;
31.
32.  public void save_to(String file) throws IOException;
```

4.3.3. Interface Médico

A interface médico, tal como as restantes, mostra agora o que um médico poderá fazer na clínica. Este pode efetuar o seu login, marcar um exame para um utente, adicionar prescrições e observações a uma determinada consulta. Pode adicionar histórico a uma ficha médica, ver a lista que possui todos os utentes, pode realizar consultas, pode ver quais as consultas de um determinado utente e as suas próprias consultas também. Além disso, pode também ver os exames marcados para um utente, a sua ficha médica tal como todas as fichas médicas dos utentes. Por fim, pode também guardar as alterações realizadas.

```java
package aula2.interfaces;

import java.io.IOException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import aula2.progs.*;
import aula2.processoclinico.*;

public interface MedicoInt extends Remote{

 public String LoginMedico(String cedula, String nif ) throws RemoteException;

 public void marca_exame(UUID id_co,int idu, String tipo, String local, LocalDate data, LocalTime hora, LocalTime duracao_exame, double preco, boolean estado, ArrayList<String> observacoes) throws RemoteException;

 public void adicionar_prescricoes(UUID id_co,LocalDate data, String toma,String dci, String nome, String formafarmaceutica, String dosagem, String estadoautorizacao, Boolean generico, String titular_aim) throws RemoteException;

 public void verlistaUtentes() throws RemoteException;

 public void adicionarObservacoes(UUID id_co,String observacoes) throws RemoteException;

 public UUID realizarConsulta(String numu, LocalDate data,LocalTime hora, double peso, int altura, double glicemia, int tensaoarterial, int colesterol, int triglicerideos, int saturacao, int inr) throws RemoteException;

 public void consultas_medico(String nummedico) throws RemoteException;

 public void adicionar_historico(String historico, String numutente) throws RemoteException;

 public void ver_fichaMedica(String numutente) throws RemoteException;

 public void exames_utente(String numutente) throws RemoteException;
 public void consultas_utente(String numutente) throws RemoteException;
 public Map<String, FichaMedica> getFichasUtente() throws RemoteException;

 public void save_to(String file) throws IOException;
```

4.3.4. Interface Gestor

A interface gestor mostra quais as ações que o gestor pode realizar. Este pode efetuar o seu login, pode adicionar um funcionário, um médico e um medicamento à base de dados da clínica. Pode ver a lista que contém os utentes, os funcionários e os médicos da clínica. Pode também ver os utentes que nasceram num determinado ano, obter a agenda que contém as consultas agendadas, as consultas realizadas na clínica, criar observações, prescrições e consultas para efetuar o carregamento inicial de dados e pode guardar as alterações efetuadas.

```java
package aula2.interfaces;

import java.io.IOException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.*;
import aula2.exceptions.MedicamentoNaoExisteException;
import aula2.exceptions.MedicoNaoExisteException;
import aula2.exceptions.UtenteNaoExisteException;
import aula2.processoclinico.Utente;
import aula2.progs.*;
import aula2.processoclinico.*;

public interface GestorInt extends Remote{

 public boolean LoginGestor(int numero, String passw ) throws RemoteException;

 public void AdicionaFunc(String nome, String morada, String nif, String cc, LocalDate datanasc, int numfunc) throws RemoteException;

 public void AdicionaMed(String nome, String morada, String nif, String cc, LocalDate datanasc, String cedula, String especialidade) throws RemoteException;

 public void AdicionaMec(String contador, String dci, String nome,
String formafarmaceutica, String dosagem, String estadoautorizacao, boolean generico,
String titular_aim) throws RemoteException;

 public void verlistaUtentes() throws RemoteException;

 public void verlistaMedicos() throws RemoteException;
 public Medico getMedico_name(String name) throws DoesNotExistsException,
RemoteException;
 public Medico getMedico_nif(String nif) throws DoesNotExistsException,RemoteException;
 public Medico getMedico_cedula(String cedula) throws DoesNotExistsException,
RemoteException;

 public void verlistaFuncionarios() throws RemoteException;
 public Funcionario getFuncionario_name(String name) throws DoesNotExistsException,
RemoteException;
 public Funcionario getFuncionario_numfunc(String numfunc) throws
DoesNotExistsException,RemoteException;
 public Funcionario getFuncionario_nif(String nif) throws DoesNotExistsException,
RemoteException;

 public void utentes_idade(int ano) throws RemoteException;
 public void save_to(String file) throws IOException;

 public Set<EntradaAgenda> getAgenda() throws RemoteException;

 public Map<UUID, Consulta> getConsultas() throws RemoteException;
 public Map<String, Medico> getMedicos() throws RemoteException;
 public Map<String, Medicamento> getMedicamentos() throws RemoteException;
 public Map<Integer, Funcionario> getFuncionarios() throws RemoteException;

 public void criar_Observacoes(Consulta co, String observacoes) throws
RemoteException;

 public Consulta adicionar_consulta(Medico med, String numutente, String Observacoes)
throws RemoteException;

 public void criar_prescricoes(Consulta co, String numutente,LocalDate data, Medicamento
medc, String toma) throws RemoteException;
}
```

4.4. Servidor

O servidor possibilita que outros computadores possam efetuar solicitações e obter como resposta o conteúdo que requisitaram. Este utiliza recurso RMI e as interfaces já implementadas e mencionadas anteriormente. O código elaborado para implementar o servidor "alves.LDA" foi o seguinte:

```
1.  package aula2.progs;
2.  import aula2.interfaces.*;
3.  import aula2.progs.PCmanager;
4.  import java.io.IOException;
5.  import java.rmi.Remote;
6.  import java.rmi.RemoteException;
7.  import java.rmi.registry.LocateRegistry;
8.  import java.rmi.registry.Registry;
9.  import java.rmi.server.RemoteStub;
10. import java.rmi.server.UnicastRemoteObject;
11.
12. public class Server implements java.io.Serializable{
13.     public static void main(String[] args) {
14.         System.setProperty("java.rmi.server.hostname", "127.0.0.1");
15.
16.         String name = "alves.LDA";
17.         PCmanager pc = null;
18.
19.         try {
20.             pc = new PCmanager();
21.             //PCmanager gestor = pc.load_from("fc.dat");
22.
23.             Registry registry = null;
24.             //LocateRegistry.createRegistry(1099);
25.
26.             registry = LocateRegistry.getRegistry();
27.
28.              Remote stub = UnicastRemoteObject.exportObject(pc,0);
29.
30.             GestorInt gestorInt = (GestorInt) stub;
31.
32.             MedicoInt med = (MedicoInt) stub;
33.
34.             UtenteInt u = (UtenteInt) stub;
35.
36.             FuncionarioInt func = (FuncionarioInt) stub;
37.
38.             registry.rebind(name, med);
39.             registry.rebind(name, gestorInt);
40.             registry.rebind(name, u);
41.             registry.rebind(name, func);
42.
43.             System.out.println("Arranquei o Servidor");
44.
45.         } catch (RemoteException e) {
46.             e.printStackTrace();
47.         } catch (IOException e) {
48.             e.printStackTrace();
49.         }
50.     }
51. }
```

4.5. Carregamento de dados

4.5.1. Carregamento dos utentes

A classe “CarregaUtentes” permite realizar o carregamento inicial dos utentes que é realizado pelo funcionário.

```
1.  package aula2.progs;
2.  import aula2.interfaces.*;
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.rmi.NotBoundException;
8.  import java.rmi.RemoteException;
9.  import java.rmi.registry.LocateRegistry;
10. import java.rmi.registry.Registry;
11. import java.time.LocalDate;
12.
13. public class CarregaUtentes {
14.     public static void main(String[] args) throws RemoteException {
15.         try {
16.             Registry reg = LocateRegistry.getRegistry("Localhost",1099);
17.             FuncionarioInt f = (FuncionarioInt) reg.lookup("alves.LDA");
18.             try {
19.                 BufferedReader br = null;
20.                 br = new BufferedReader(new FileReader("/Users/joaom/uminho/aplicações
    distribuídas/TP_1/src/aula2/ficheiros/utentes.txt"));
21.                 String line;
22.                 while ((line = br.readLine()) != null) {
23.                     String [] tokens = line.split(";");
24.                     for (int i = 0; i < tokens.length; i++) {
25.                         if (tokens[i].startsWith("\"")) {
26.                             tokens[i] = tokens[i].replace('"', ' ').strip();
27.                         }
28.                     }
29.                     String[] dn = tokens[4].split("-");
30.                     f.AdicionaUtente(tokens[0], tokens[1], tokens[2],
31.                             tokens[3], LocalDate.of(
32.                                     Integer.parseInt(dn[0]),
33.                                     Integer.parseInt(dn[1]),
34.                                     Integer.parseInt(dn[2])
35.                             ),
36.                             tokens[5], tokens[6], tokens[7], tokens[8]);
37.                 }
38.             f.save_to("fc.dat");
39.             System.out.println("Utentes adicionados!");
40.
41.             } catch (FileNotFoundException e) {
42.                 e.printStackTrace();
43.             } catch (IOException e) {
44.                 e.printStackTrace();
45.             }
46.
47.         } catch (
48.                 RemoteException remoteException) {
49.             remoteException.printStackTrace();
50.         } catch (NotBoundException notBoundException) {
51.             notBoundException.printStackTrace();
52.         }
```

4.5.2. Carregamento dos funcionários

A classe “CarregaFuncionarios” permite realizar o carregamento inicial dos funcionários, cuja realização é feita pelo gestor da clínica.

```
1.  package aula2.progs;
2.  import aula2.interfaces.*;
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.rmi.NotBoundException;
8.  import java.rmi.RemoteException;
9.  import java.rmi.registry.LocateRegistry;
10. import java.rmi.registry.Registry;
11. import java.time.LocalDate;
12.
13. public class CarregaFuncionarios {
14.     public static void main(String[] args) throws RemoteException {
15.         try {
16.
17.             Registry reg = LocateRegistry.getRegistry("Localhost",1099);
18.             GestorInt g = (GestorInt) reg.lookup("alves.LDA");
19.             BufferedReader br = null;
20.             try {
21.                 br = new BufferedReader(new FileReader("/Users/joaom/uminho/aplicações
    distribuídas/TP_1/src/aula2/ficheiros/funcionarios.txt"));
22.                 String line;
23.                 while ((line = br.readLine()) != null) {
24.                     String []tokens = line.split(";");
25.                     for (int i = 0; i < tokens.length; i++) {
26.                         if (tokens[i].startsWith("\"")) {
27.                             tokens[i] = tokens[i].replace('"', ' ').strip();
28.                         }
29.                     }
30.                     String[] dn = tokens[4].split("-");
31.                     g.AdicionaFunc(tokens[0], tokens[1], tokens[2],
32.                             tokens[3], LocalDate.of(
33.                                     Integer.parseInt(dn[0]),
34.                                     Integer.parseInt(dn[1]),
35.                                     Integer.parseInt(dn[2])),
36.                             Integer.parseInt(tokens[5]));
37.                 }
38.             g.save_to("fc.dat");
39.             System.out.println("Funcionarios adicionados!");
40.             } catch (FileNotFoundException e) {
41.                 e.printStackTrace();
42.             } catch (IOException e) {
43.                 e.printStackTrace();
44.             }
45.
46.         } catch (RemoteException remoteException) {
47.             remoteException.printStackTrace();
48.         } catch (NotBoundException notBoundException) {
49.             notBoundException.printStackTrace();
50.         }
51.
52.  }
```

4.5.3. Carregamento dos medicamentos

A classe "CarregaMedicamentos" permite efetuar o carregamento inicial dos medicamentos. Esta ação é realizada também pelo gestor da clínica.

```
1.  package aula2.progs;
2.  import aula2.interfaces.*;
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.rmi.NotBoundException;
8.  import java.rmi.RemoteException;
9.  import java.rmi.registry.LocateRegistry;
10. import java.rmi.registry.Registry;
11.
12. public class CarregaMedicamentos {
13.     public static void main(String[] args) throws RemoteException {
14.         try {
15.
16.             Registry reg = LocateRegistry.getRegistry("Localhost", 1099);
17.             GestorInt g = (GestorInt) reg.lookup("alves.LDA");
18.             BufferedReader br = null;
19.
20.             try {
21.                 br = new BufferedReader(new FileReader("/Users/joaom/uminho/aplicações
    distribuídas/TP_1/src/aula2/ficheiros/lista_infomed_2020.csv"));
22.                 String line;
23.                 while ((line = br.readLine()) != null) {
24.                     String[] tokens = line.split(";");
25.                     System.out.println(tokens[0]);
26.                     g.AdicionaMec(tokens[0],tokens[1], tokens[2],
27.                             tokens[3], tokens[4],
28.                             tokens[5], Boolean.getBoolean(tokens[6]),
29.                             tokens[7]
30.                     );
31.                 }
32.             } catch (FileNotFoundException e) {
33.                 e.printStackTrace();
34.             } catch (IOException e) {
35.                 e.printStackTrace();
36.             }
37.             g.save_to("fc.dat");
38.             System.out.println("Medicamentos adicionados!");
39.         } catch (RemoteException remoteException) {
40.             remoteException.printStackTrace();
41.         } catch (NotBoundException notBoundException) {
42.             notBoundException.printStackTrace();
43.         } catch (IOException e) {
44.             e.printStackTrace();
45.         }
46.     }
47. }
```

4.5.4. Carregamento dos médicos

A classe "CarregaMedicos" permite fazer o carregamento inicial dos médicos da clínica. Esta ação é feita novamente pelo gestor da clínica.

```java
package aula2.progs;
import aula2.interfaces.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.time.LocalDate;

public class CarregaMedicos {
    public static void main(String[] args) throws RemoteException {


        try {

            Registry reg = LocateRegistry.getRegistry("Localhost",1099);
            GestorInt g = (GestorInt) reg.lookup("alves.LDA");
            BufferedReader br = null;
            try {
                br = new BufferedReader(new FileReader("/Users/joaom/uminho/aplicações distribuídas/TP_1/src/aula2/ficheiros/medicos.txt"));
                String line;
                while ((line = br.readLine()) != null) {
                    String []tokens = line.split(";");
                    for (int i = 0; i < tokens.length; i++) {
                        if (tokens[i].startsWith("\"")) {
                            tokens[i] = tokens[i].replace('"', ' ').strip();
                        }
                    }
                    String[] dn = tokens[4].split("-");
                    g.AdicionaMed(tokens[0], tokens[1], tokens[2],
                            tokens[3], LocalDate.of(Integer.valueOf(dn[0]),
                                    Integer.valueOf(dn[1]),
                                    Integer.valueOf(dn[2])),
                            tokens[5], tokens[6]);
                }
            g.save_to("fc.dat");
            System.out.println("Medicos adicionados!");

            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (RemoteException remoteException) {
            remoteException.printStackTrace();
        } catch (NotBoundException notBoundException) {
            notBoundException.printStackTrace();
        }
    }
}
```

4.5.5. Carregamento das consultas e do histórico

A classe "CarregaHistoricoeConsultas" permite realizar o carregamento inicial das consultas e histórico. Para tal é necessário um gestor e um médico.

```java
package aula2.progs;
import aula2.interfaces.*;
import aula2.exceptions.MedicamentoNaoExisteException;
import aula2.exceptions.MedicoNaoExisteException;
import aula2.exceptions.UtenteNaoExisteException;
import aula2.processoclinico.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.time.LocalDate;
import java.util.Map;

public class CarregaHistoricoeConsultas {
    public static void main(String[] args) throws RemoteException {
        try {
            Registry reg = LocateRegistry.getRegistry("Localhost",1099);
            MedicoInt medico = (MedicoInt) reg.lookup("alves.LDA");
            GestorInt g = (GestorInt) reg.lookup("alves.LDA");
            BufferedReader br = null;
            try {
                br = new BufferedReader(new FileReader("/Users/joaom/uminho/aplicações distribuídas/TP_1/src/aula2/ficheiros/fichautentes.txt"));
                String line;
                Map<String, FichaMedica> fichasUtente = medico.getFichasUtente();
                while ((line = br.readLine()) != null) {
                    String []tokens = line.split(";");
                    for (int i = 0; i < tokens.length; i++) {
                        if (tokens[i].startsWith("\"")) {
                            tokens[i] = tokens[i].replace('"', ' ').strip();
                        }
                    }
                    //if (!fichasUtente.containsKey(tokens[0])) {
                        //throw new UtenteNaoExisteException();
                    //}
                    medico.adicionar_historico(tokens[1], tokens[0]);
                }
                br.close();
                medico.save_to("fc.dat");

                Map<String, Medico> medicos = g.getMedicos();
                Map<String, Medicamento> medicamentos = g.getMedicamentos();
                br = new BufferedReader( new FileReader("/Users/joaom/uminho/aplicações distribuídas/TP_1/src/aula2/ficheiros/consultas.txt"));
                while( (line = br.readLine()) != null){
                    String []tokens = line.split(";");
                    for(int i=0 ; i < tokens.length; i++){
                        if (tokens[i].startsWith("\"")){
                            tokens[i] = tokens[i].replace('"',' ').strip();
                        }
                    }
                    String []data = tokens[2].split("-");

                    //if (! medicos.containsKey(tokens[0])){
                        //throw new MedicoNaoExisteException();
                    //}

                    Medico m = medicos.get(tokens[0]);

                    //if (! fichasUtente.containsKey(tokens[1])){
                        //throw  new UtenteNaoExisteException();
                    //}
```

```
65.                     Consulta co = g.adicionar_consulta(m, tokens[1], tokens[3]);
66.
67.                     for(int i = 4; i < tokens.length; i++){
68.                         String token = tokens[i];
69.                         token = token.
70.                                 replace('[',' ').
71.                                 replace( ']',' ').
72.                                 strip();
73.                         String[] rtokens = token.split(",");
74.                         for(int j=0 ; j < rtokens.length; j++){
75.                             rtokens[j] = rtokens[j].replace('\'',' ').strip();
76.                         }
77.                         data = rtokens[0].split("-");
78.
79.                         //if(! medicamentos.containsKey(rtokens[1])){
80.                             //throw new MedicamentoNaoExisteException();
81.                         //}
82.                         Medicamento medc = medicamentos.get(rtokens[1]);
83.                         g.criar_prescricoes(co,tokens[1],LocalDate.of(
84.                                 Integer.parseInt(data[0]),
85.                                 Integer.parseInt(data[1]),
86.                                 Integer.parseInt(data[2])),
87.                                 medc, rtokens[2]);
88.                     }
89.                 }
90.                 g.save_to("fc.dat");
91.                 System.out.println("Consultas e fichas utentes adicionadas!");
92.
93.             } catch (FileNotFoundException e) {
94.                 e.printStackTrace();
95.             } catch (IOException e) {
96.                 e.printStackTrace();
97.             } //catch (MedicoNaoExisteException e) {
98.                 //e.printStackTrace();
99.              //catch (MedicamentoNaoExisteException e) {
100.                //e.printStackTrace();
101.            //} //catch (UtenteNaoExisteException e) {
102.                //e.printStackTrace();
103.
104.        } catch (RemoteException remoteException) {
105.            remoteException.printStackTrace();
106.        } catch (NotBoundException notBoundException) {
107.            notBoundException.printStackTrace();
108.        }
109.    }
110.}
```

4.6. Cliente

O cliente executa todos os recursos e métodos implementados.

No cliente, existem várias funcionalidades que serão executadas e para cada uma delas, há uma conexão correspondente à interface. Este irá receber pedidos do gestor, médico, utente ou funcionário e irá executar os comandos necessários para fornecer dados que o servidor usará para responder aos pedidos realizados. No código a seguir estão exemplificados alguns desses comandos.

```java
package aula2.progs;
import aula2.interfaces.*;
import aula2.exceptions.MedicamentoNaoExisteException;
import aula2.exceptions.MedicoNaoExisteException;
import aula2.exceptions.UtenteNaoExisteException;
import aula2.processoclinico.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Map;
import java.util.UUID;

public class Cliente {
    public static void main(String[] args) {
        try {
            Registry reg = LocateRegistry.getRegistry("Localhost",1099);
            GestorInt g = (GestorInt) reg.lookup("alves.LDA");
            FuncionarioInt f = (FuncionarioInt) reg.lookup("alves.LDA");
            UtenteInt u = (UtenteInt) reg.lookup("alves.LDA");
            MedicoInt medico = (MedicoInt) reg.lookup("alves.LDA");
            System.out.println(g.getMedicamentos());
            //System.out.println(medico.getFichasUtente().size());
            //f.AdicionaUtente("Joaquina Alves","Rua da Luz","123456789","88888888 ZYX",
 LocalDate.of(1931,10,29),"1234","939255568","929578854","joaquinaalves@gmail.com");
            //System.out.println(u.LoginUtente("1234","123456789"));
            //System.out.println(medico.LoginMedico("0975-351","PT578156593"));
            //System.out.println(f.LoginFunc(1,"PT675375100"));
            //System.out.println(u.marca_consulta("1234",LocalDate.of(2020,11,29),
LocalTime.of(14,00),"Urologia"));
            //System.out.println(u.marca_consulta("1234",LocalDate.of(2020,11,29),
LocalTime.of(14,35),"Urologia"));
            //u.cancelar_consulta("1234", LocalDate.of(2020,11,29),LocalTime.of(14,00),
"Urologia");
            //UUID num = medico.realizarConsulta("1234",LocalDate.of(2020,11,29),
LocalTime.of(14, 00), 60.00,160, 1.00, 200, 300, 250, 1, 12);
            //medico.adicionarObservacoes(num,"Ta bom de saude!");
            //medico.adicionar_prescricoes(num, LocalDate.of(2020,11,29), "2 vezes",
"Abacavir","Ziagen","Comprimido revestido por película","300 mg","Autorizado",false,
"ViiV Healthcare UK Ltd.");
            // medico.adicionar_historico("es grande","1234");

            //medico.consultas_utente("1234");

            //medico.marca_exame(num,1234,"olhos","braga",LocalDate.of(2020,12,29), Loca
lTime.of(14, 00), LocalTime.of(00, 20),50.50,true, new ArrayList<String>());

            //System.out.println(g.getConsultas());
            //medico.ver_fichaMedica("1234");
            //g.utentes_idade(1931);
            //System.out.println(medico.getFichasUtente());
            //System.out.println(g.getFuncionarios());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5. Conclusão

O presente trabalho permitiu a familiarização com a linguagem de programação Java, uma vez que as bases provenientes de anos anteriores eram escassas. Para tal foi desenvolvida e implementada uma arquitetura cliente-servidor.

Para a realização desta arquitetura recorreu-se ao RMI para que se pudesse alcançar o objetivo, que era implementar um servidor, o qual diferentes clientes pudessem aceder remotamente, realizar pedidos e obter respostas a esses mesmo pedidos.

Desta forma, foi possível manipular vários processos por vários utilizadores, correspondentes a diferentes entidades (utente, médico, funcionário e gestor).

O trabalho e, consequentemente, os seus resultados e os requisitos pedidos para a elaboração do projeto foram alcançados, apesar das eventuais dificuldades que foram aparecendo e que tiveram continuamente de ser ultrapassadas.

Concluindo, sugere-se que, no futuro, haja uma possível melhoria de forma a aperfeiçoar o desempenho da aplicação implementada. Para tal, poder-se-ia acrescentar novas funcionalidades, métodos que pudessem refletir o pedido de espera quando se tira uma senha numa clínica ou hospital e outras entidades tais como secretárias e/ou técnicos que também se podem encontrar numa clínica ou hospital.

6. Bibliografia

[1] https://www.inf.ufsc.br/~frank.siqueira/INE5645/3.%20Controle%20de%20Concorrencia.pdf, consultado em 27 novembro, 2020;

[2] https://web.fe.up.pt/~eol/AIAD/aulas/JINIdocs/rmi1.html, consultado em 1 dezembro, 2020;

[3] https://sites.google.com/site/proffdesiqsistemasdistribuidos/aulas/caracterizacao-de-sistemas-distribuidos, consultado em 1 dezembro, 2020;

[4] https://web.fe.up.pt/~goii2000/M1/2_1-clienteservidor.htm, consultado em 28 novembro 2020;

[5] COULOURIS G.; DOLLIMORE J.;KINDBERG T; (2007) Sistemas Distribuídos: Conceitos e Projeto. Bookman 4ª Edição. Consultado em 28 novembro,2020. Disponível em: https://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/coulouris.pdf

7. Anexos

A. CLASSES (classes implementadas que não foram referidas no ponto 4.1.)

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Medico extends Pessoa implements Serializable {
7.
8.      private String cedula;
9.      private String especialidade;
10.
11.     public Medico(String cedula, String especialidade) {
12.         this.cedula = cedula;
13.         this.especialidade = especialidade;
14.     }
15.
16.     public Medico(String nome, String morada, String nif, String cc, LocalDate datanasc,
     String cedula, String especialidade) {
17.         super(nome, morada, nif, cc, datanasc);
18.         this.cedula = cedula;
19.         this.especialidade = especialidade;
20.     }
21.
22.     public String getCedula() {
23.         return cedula;
24.     }
25.
26.     public void setCedula(String cedula) {
27.         this.cedula = cedula;
28.     }
29.
30.     public String getEspecialidade() {
31.         return especialidade;
32.     }
33.
34.     public void setEspecialidade(String especialidade) {
35.         this.especialidade = especialidade;
36.     }
37.
38.     @Override
39.     public String toString() {
40.         return "Medico{" +
41.                 super.toString() +
42.                 "cedula='" + cedula + '\'' +
43.                 ", especialidade='" + especialidade + '\'' +
44.                 '}';
45.     }
46. }
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Funcionario extends Pessoa implements Serializable {
7.
```

```
8.      private int numfunc;
9.
10.     public Funcionario(int numfunc) {
11.         this.numfunc = numfunc;
12.     }
13.
14.     public Funcionario(String nome, String morada, String nif, String cc, LocalDate data
    nasc, int numfunc) {
15.         super(nome, morada, nif, cc, datanasc);
16.         this.numfunc = numfunc;
17.     }
18.
19.     public int getNumfunc() {
20.         return numfunc;
21.     }
22.
23.     public void setNumfunc(int numfunc) {
24.         this.numfunc = numfunc;
25.     }
26.
27.     @Override
28.     public String toString() {
29.         return "Funcionario{" +
30.                 super.toString()+
31.                 "numfunc=" + numfunc +
32.                 '}';
33.     }
34. }
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Gestor extends Pessoa implements Serializable{
7.
8.      private String telefone;
9.      private String telefone_emergencia;
10.     private String email;
11.
12.     public Gestor(String telefone, String telefone_emergencia, String email) {
13.         this.telefone = telefone;
14.         this.telefone_emergencia = telefone_emergencia;
15.         this.email = email;
16.     }
17.
18.     public Gestor(String nome, String morada, String nif, String cc, LocalDate datanasc,
     String telefone, String telefone_emergencia, String email) {
19.         super(nome, morada, nif, cc, datanasc);
20.         this.telefone = telefone;
21.         this.telefone_emergencia = telefone_emergencia;
22.         this.email = email;
23.     }
24.
25.     public String getTelefone() {
26.         return telefone;
27.     }
28.     public void setTelefone(String telefone) {
29.         this.telefone = telefone;
30.     }
31.
32.     public String getTelefone_emergencia() {
33.         return telefone_emergencia;
34.     }
```

```
35.     public void setTelefone_emergencia(String telefone_emergencia) {
36.         this.telefone_emergencia = telefone_emergencia;
37.     }
38.
39.     public String getEmail() {
40.         return email;
41.     }
42.
43.     public void setEmail(String email) {
44.         this.email = email;
45.     }
46.
47.     @Override
48.     public String toString() {
49.         return "Gestor{" +
50.                 "telefone='" + telefone + '\'' +
51.                 ", telefone_emergencia='" + telefone_emergencia + '\'' +
52.                 ", email='" + email + '\'' +
53.                 '}';
54.     }
55. }
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  import java.util.ArrayList;
7.  import java.util.List;
8.
9.  public class Consulta implements Serializable {
10.
11.
12.     private LocalDate data;
13.     private String observacoes;
14.     private List<Prescricao> prescricoes;
15.     private List<Exame> exames;
16.     private Medico medico;
17.
18.     public Consulta(LocalDate data) {
19.         this.data = data;
20.         this.observacoes="";
21.         this.prescricoes = new ArrayList<>();
22.         this.exames  = new ArrayList<>();
23.     }
24.
25.     public Consulta(LocalDate data, String observacoes, List<Prescricao> prescricoes,
    List<Exame> exames) {
26.         this.data = data;
27.         this.observacoes = observacoes;
28.         this.prescricoes = prescricoes;
29.         this.exames = exames;
30.     }
31.     public Consulta() {
32.         this(LocalDate.now());
33.     }
34.     public Consulta(Medico m) {
35.         this(LocalDate.now());
36.         this.medico = m;
37.     }
38.
39.     public void setMedico(Medico medico) {
40.         this.medico = medico;
41.     }
```

```
42.
43.     public LocalDate getData() {
44.         return data;
45.     }
46.
47.     public String getObservacoes() {
48.         return observacoes;
49.     }
50.
51.     public List<Prescricao> getPrescricoes() {
52.         return prescricoes;
53.     }
54.
55.     public List<Exame> getExames() {
56.         return exames;
57.     }
58.     public void addExame(Exame exame) {
59.         this.exames.add(exame);
60.     }
61.
62.     public Medico getMedico() {
63.         return medico;
64.     }
65.
66.     public void addObservacao(String obs){
67.         this.observacoes = this.observacoes.concat("\n"+obs);
68.     }
69.
70.     public void addPrescricao(Prescricao p){
71.         this.prescricoes.add(p);
72.     }
73.
74.     @Override
75.     public String toString() {
76.         return "Consulta{" +
77.                 "data=" + data +
78.                 ", observacoes='" + observacoes + '\'' +
79.                 ", prescricoes=" + prescricoes +
80.                 ", exames=" + exames +
81.                 ", medico=" + medico +
82.                 '}';
83.     }
84. }
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Medicao implements Serializable {
7.
8.      private LocalDate data;
9.      private double peso;
10.     private int altura;
11.     private double glicemia;
12.     private int tensaoarterial;
13.     private int colesterol;
14.     private int triglicerideos;
15.     private int saturacao;
16.     private int inr;
17.
18.     public Medicao() {
19.         data = LocalDate.now();
20.     }
```

```
21.     public Medicao(LocalDate data, double peso, int altura, double glicemia, int
    tensaoarterial, int colesterol, int triglicerideos, int saturacao, int inr) {
22.         this.data = data;
23.         this.peso = peso;
24.         this.altura = altura;
25.         this.glicemia = glicemia;
26.         this.tensaoarterial = tensaoarterial;
27.         this.colesterol = colesterol;
28.         this.triglicerideos = triglicerideos;
29.         this.saturacao = saturacao;
30.         this.inr = inr;
31.     }
32.
33.     public double getPeso() {
34.         return peso;
35.     }
36.
37.     public void setPeso(double peso) {
38.         this.peso = peso;
39.     }
40.
41.     public int getAltura() {
42.         return altura;
43.     }
44.
45.     public void setAltura(int altura) {
46.         this.altura = altura;
47.     }
48.
49.     public double getGlicemia() {
50.         return glicemia;
51.     }
52.
53.     public void setGlicemia(double glicemia) {
54.         this.glicemia = glicemia;
55.     }
56.
57.     public int getTensaoarterial() {
58.         return tensaoarterial;
59.     }
60.
61.     public void setTensaoarterial(int tensaoarterial) {
62.         this.tensaoarterial = tensaoarterial;
63.     }
64.
65.     public int getColesterol() {
66.         return colesterol;
67.     }
68.
69.     public void setColesterol(int colesterol) {
70.         this.colesterol = colesterol;
71.     }
72.
73.     public int getTriglicerideos() {
74.         return triglicerideos;
75.     }
76.
77.     public void setTriglicerideos(int triglicerideos) {
78.         this.triglicerideos = triglicerideos;
79.     }
80.
81.     public int getSaturacao() {
82.         return saturacao;
83.     }
84.
85.     public void setSaturacao(int saturacao) {
```

```
86.         this.saturacao = saturacao;
87.     }
88.
89.     public int getInr() {
90.         return inr;
91.     }
92.
93.     public void setInr(int inr) {
94.         this.inr = inr;
95.     }
96.
97.     @Override
98.     public String toString() {
99.         return "Medicao{" +
100.                "data=" + data +
101.                ", peso=" + peso +
102.                ", altura=" + altura +
103.                ", glicemia=" + glicemia +
104.                ", tensaoarterial=" + tensaoarterial +
105.                ", colesterol=" + colesterol +
106.                ", triglicerideos=" + triglicerideos +
107.                ", saturacao=" + saturacao +
108.                ", inr=" + inr +
109.                '}';
110.    }
111.}
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.rmi.RemoteException;
5.  import java.rmi.server.UnicastRemoteObject;
6.  import java.time.LocalDate;
7.  import java.time.LocalTime;
8.  import java.util.ArrayList;
9.  import java.util.List;
10.
11. public class Exame implements Serializable {
12.
13.     private static int id_exame;
14.     public int idu;
15.     public String tipo;
16.     public String local;
17.     public LocalDate data;
18.     public LocalTime hora;
19.     public LocalTime duracao_exame;
20.     public double preco;
21.     public boolean estado;
22.     public List observacoes;
23.
24.     public Exame(int idu, String tipo, String local, LocalDate data, LocalTime hora,
    LocalTime duracao_exame, double preco, boolean estado, ArrayList<String> observacoes) {
25.         this.idu = idu;
26.         this.tipo = tipo;
27.         this.local = local;
28.         this.data = data;
29.         this.hora = hora;
30.         this.duracao_exame = duracao_exame;
31.         this.preco = preco;
32.         this.estado = estado;
33.         this.observacoes = observacoes;
34.     }
35.     public Exame(int idu, String tipo, String local, LocalDate data, LocalTime hora) thr
    ows RemoteException {
```

```
36.         this.idu = idu;
37.         this.tipo = tipo;
38.         this.local = local;
39.         this.data=data;
40.         this.hora=hora;
41.     }
42.
43.     public static int getId_exame() {
44.         return id_exame;
45.     }
46.
47.     public static void setId_exame(int id_exame) {
48.         Exame.id_exame = id_exame;
49.     }
50.
51.     public int getIdu() {
52.         return idu;
53.     }
54.
55.     public void setIdu(int idu) {
56.         this.idu = idu;
57.     }
58.
59.     public String getTipo() {
60.         return tipo;
61.     }
62.
63.     public void setTipo(String tipo) {
64.         this.tipo = tipo;
65.     }
66.
67.     public String getLocal() {
68.         return local;
69.     }
70.
71.     public void setLocal(String local) {
72.         this.local = local;
73.     }
74.
75.     public LocalDate getData() {
76.         return data;
77.     }
78.
79.     public void setData(LocalDate data) {
80.         this.data = data;
81.     }
82.
83.     public LocalTime getHora() {
84.         return hora;
85.     }
86.
87.     public void setHora(LocalTime hora) {
88.         this.hora = hora;
89.     }
90.
91.     public LocalTime getDuracao_exame() {
92.         return duracao_exame;
93.     }
94.
95.     public void setDuracao_exame(LocalTime duracao_exame) {
96.         this.duracao_exame = duracao_exame;
97.     }
98.
99.     public double getPreco() {
100.        return preco;
101.    }
```

```
102.
103.    public void setPreco(double preco) {
104.        this.preco = preco;
105.    }
106.
107.    public boolean isEstado() {
108.        return estado;
109.    }
110.
111.    public void setEstado(boolean estado) {
112.        this.estado = estado;
113.    }
114.
115.    public List getObservacoes() {
116.        return observacoes;
117.    }
118.
119.    public void setObservacoes(List observacoes) {
120.        this.observacoes = observacoes;
121.    }
122.
123.    @Override
124.    public String toString() {
125.        return "Exame{" +
126.                "idu=" + idu +
127.                ", tipo='" + tipo + '\'' +
128.                ", local='" + local + '\'' +
129.                ", data=" + data +
130.                ", hora=" + hora +
131.                ", duracao_exame=" + duracao_exame +
132.                ", preco=" + preco +
133.                ", estado=" + estado +
134.                ", observacoes=" + observacoes +
135.                '}';
136.    }
137.}
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.
5.  public class Medicamento implements Serializable {
6.
7.      private String dci;
8.      private String nome;
9.      private String formafarmaceutica;
10.     private String dosagem;
11.     private String estadoautorizacao;
12.     private boolean generico;
13.     private String titular_aim;
14.
15.     public Medicamento(String dci) {
16.         this.dci = dci;
17.     }
18.     public Medicamento(String dci, String nome, String formafarmaceutica, String dosagem
    , String estadoautorizacao, boolean generico, String titular_aim) {
19.         this.dci = dci;
20.         this.nome = nome;
21.         this.formafarmaceutica = formafarmaceutica;
22.         this.dosagem = dosagem;
23.         this.estadoautorizacao = estadoautorizacao;
24.         this.generico = generico;
25.         this.titular_aim = titular_aim;
26.     }
```

```
27.
28.     public String getDci() {
29.         return dci;
30.     }
31.
32.     public String getNome() {
33.         return nome;
34.     }
35.
36.     public String getFormafarmaceutica() {
37.         return formafarmaceutica;
38.     }
39.
40.     public String getDosagem() {
41.         return dosagem;
42.     }
43.
44.     public String getEstadoautorizacao() {
45.         return estadoautorizacao;
46.     }
47.
48.     public boolean isGenerico() {
49.         return generico;
50.     }
51.
52.     public String getTitular_aim() {
53.         return titular_aim;
54.     }
55.
56.     @Override
57.     public String toString() {
58.         return "Medicamento{" +
59.                 "dci='" + dci + '\'' +
60.                 ", nome='" + nome + '\'' +
61.                 ", formafarmaceutica='" + formafarmaceutica + '\'' +
62.                 ", dosagem='" + dosagem + '\'' +
63.                 ", estadoautorizacao='" + estadoautorizacao + '\'' +
64.                 ", generico=" + generico +
65.                 ", titular_aim='" + titular_aim + '\'' +
66.                 '}';
67.     }
68. }
```

```
1.  package aula2.processoclinico;
2.
3.  import java.io.Serializable;
4.  import java.time.LocalDate;
5.
6.  public class Prescricao implements Serializable {
7.
8.      private LocalDate data;
9.      private Medicamento med;
10.     private String toma;
11.
12.     public Prescricao() {
13.         data = LocalDate.now();
14.     }
15.
16.     public Prescricao(LocalDate data, Medicamento med, String toma) {
17.         this.data = data;
18.         this.med = med;
19.         this.toma = toma;
20.     }
21.
```

```
22.     public Medicamento getMed() {
23.         return med;
24.     }
25.
26.     public void setMed(Medicamento med) {
27.         this.med = med;
28.     }
29.
30.     public String getToma() {
31.         return toma;
32.     }
33.
34.     public void setToma(String toma) {
35.         this.toma = toma;
36.     }
37.
38.     @Override
39.     public String toString() {
40.         return "Prescricao{" +
41.                 "data=" + data +
42.                 ", med=" + med +
43.                 ", toma='" + toma + '\'' +
44.                 '}';
45.     }
46. }
```

```
1.  package aula2.processoclinico;
2.  import java.io.Serializable;
3.  import java.time.LocalDate;
4.  import java.time.LocalDateTime;
5.  import java.time.LocalTime;
6.  import java.util.List;
7.  import java.util.Map;
8.  import java.util.ArrayList;
9.
10. public class EntradaAgenda implements Serializable {
11.
12.     private LocalTime hora;
13.     private LocalDate data;
14.     private LocalTime duracao = LocalTime.parse("00:30:00");
15.     private Medico medico;
16.     private Utente utente;
17.     private String estado;
18.
19.     public EntradaAgenda(LocalTime hora, LocalDate data, Medico medico, Utente utente, S
    tring estado) {
20.         this.hora = hora;
21.         this.data = data;
22.         this.medico = medico;
23.         this.utente = utente;
24.         this.estado = estado;
25.     }
26.
27.
28.     public LocalTime getHora() {
29.         return hora;
30.     }
31.
32.     public void setHora(LocalTime hora) {
33.         this.hora = hora;
34.     }
35.
36.     public LocalDate getData() {
37.         return data;
```

```
38.     }
39.
40.     public void setData(LocalDate data) {
41.         this.data = data;
42.     }
43.
44.     public LocalTime getDuracao() {
45.         return duracao;
46.     }
47.
48.     public Medico getMedico() {
49.         return medico;
50.     }
51.
52.     public void setMedico(Medico medico) {
53.         this.medico = medico;
54.     }
55.
56.     public Utente getUtente() {
57.         return utente;
58.     }
59.
60.     public void setUtente(Utente utente) {
61.         this.utente = utente;
62.     }
63.
64.     public String getEstado() {
65.         return estado;
66.     }
67.
68.     public void setEstado(String estado) {
69.         this.estado = estado;
70.     }
71.
72.     @Override
73.     public String toString() {
74.         return "EntradaAgenda{" +
75.                 "hora=" + hora +
76.                 ", data=" + data +
77.                 ", duracao=" + duracao +
78.                 ", medico=" + medico +
79.                 ", utente=" + utente +
80.                 ", estado='" + estado + '\'' +
81.                 '}';
82.     }
83. }
```

```
1.  package aula2.processoclinico;
2.  import java.io.Serializable;
3.  import java.util.Set;
4.
5.  import java.util.HashSet;
6.
7.  public class Agenda implements Serializable {
8.
9.      private Set <EntradaAgenda> agenda;
10.
11.     public Agenda(){
12.         agenda = new HashSet<>();
13.     }
14.
15.     public Set<EntradaAgenda> getAgenda() {
16.         return agenda;
17.     }
```

```
18.
19.     public void setAgenda(Set<EntradaAgenda> agenda) {
20.         this.agenda = agenda;
21.     }
22.
23.     @Override
24.     public String toString() {
25.         return "Agenda{" +
26.                 "agenda=" + agenda +
27.                 '}';
28.     }
29. }
```

B. PC MANAGER (métodos implementados não apresentados no ponto 4.2.)

```
1.  package aula2.progs;
2.
3.  import aula2.exceptions.DoesNotExistsException;
4.  import aula2.exceptions.MedicamentoNaoExisteException;
5.  import aula2.exceptions.MedicoNaoExisteException;
6.  import aula2.exceptions.UtenteNaoExisteException;
7.  import aula2.processoclinico.*;
8.  import aula2.interfaces.*;
9.  import java.rmi.Remote;
10. import java.rmi.RemoteException;
11. import java.time.LocalTime;
12. import java.io.*;
13. import java.time.LocalDate;
14. import java.util.*;
15. import java.util.UUID;
16.
17. public class PCmanager implements UtenteInt, GestorInt, MedicoInt, FuncionarioInt,
    Serializable, Remote {
18.
19.     private static final long serialVersionUID = -9149826635246618824L;
20.     private Map<String, FichaMedica> fichasUtente;
21.     private Map<String, Medico> medicos;
22.     private Map<Integer, Funcionario> funcionarios;
23.     private Map<String, Medicamento> medicamentos;
24.     public Map<Consulta, List<Exame> > exames = new HashMap<>();
25.     public Map<UUID, Consulta> consultas = new HashMap<>();
26.     public Set <EntradaAgenda> agenda = new HashSet<>();
27.     public Map<Integer, EntradaAgenda> eag =  new HashMap();
28.     private Map<String, Utente> utente_name;
29.     private Map<String, Utente> utente_nif;
30.     private Map<String, Utente> utente_nutente;
31.     private Map<String, Medico> medico_name;
32.     private Map<String, Medico> medico_nif;
33.     private Map<String, Medico> medico_cedula;
34.     private Map<String, Funcionario> funcionario_name;
35.     private Map<Integer, Funcionario> funcionario_numfunc;
36.     private Map<String, Funcionario> funcionario_nif;
37.
38.     public PCmanager() {
39.         fichasUtente = new HashMap<>();
40.         medicos = new HashMap<>();
41.         funcionarios = new HashMap<>();
42.         medicamentos = new HashMap<>();
43.     }
44.
45.
```

```
46.     @Override
47.     public void save_to(String file) throws IOException {
48.         ObjectOutputStream os = new ObjectOutputStream( new FileOutputStream(file) );
49.         os.writeObject(this);
50.         os.close();
51.     }
52.
53.     public static PCmanager load_from(String file) throws IOException,
    ClassNotFoundException {
54.         ObjectInputStream is = new ObjectInputStream( new FileInputStream(file));
55.         PCmanager o = (PCmanager) is.readObject();
56.         return o;
57.     }
58.     @Override
59.     public Map<String, FichaMedica> getFichasUtente() throws RemoteException{
60.         return fichasUtente;
61.     }
62.     @Override
63.     public Map<String, Medico> getMedicos() throws RemoteException{
64.         return medicos;
65.     }
66.
67.     public Map<Integer, Funcionario> getFuncionarios() throws RemoteException{
68.         return funcionarios;
69.     }
70.
71.       @Override
72.     public Map<String, Medicamento> getMedicamentos() throws RemoteException {
73.         return medicamentos;
74.     }
75.     @Override
76.     public Set<EntradaAgenda> getAgenda() throws RemoteException {
77.         return agenda;
78.     }
79.
80.       public Map<UUID, Consulta> getConsultas() throws RemoteException{
81.         return consultas;
82.     }
83.
84.     @Override
85.     public Utente getUtente(String nome) throws DoesNotExistsException,
    RemoteException  {
86.         if (utente_name.containsKey(nome)) {
87.             return utente_name.get(nome);
88.         } else {
89.             throw new DoesNotExistsException();
90.         }
91.     }
92.     @Override
93.     public Utente getUtente_nif(String nif) throws DoesNotExistsException,
    RemoteException {
94.         if (utente_nif.containsKey(nif)) {
95.             return utente_nif.get(nif);
96.         } else {
97.             throw new DoesNotExistsException();
98.         }
99.     }
100.    @Override
101.    public Utente getUtente_numutente(String numutente) throws DoesNotExistsException,
    RemoteException {
102.        if (utente_nutente.containsKey(numutente)) {
103.            return utente_nutente.get(numutente);
104.        } else {
105.            throw new DoesNotExistsException();
106.        }
107.    }
```

```
108.    @Override
109.    public Medico getMedico_name(String name) throws DoesNotExistsException,
    RemoteException {
110.        if (medico_name.containsKey(name)) {
111.            return medico_name.get(name);
112.        } else {
113.            throw new DoesNotExistsException();
114.        }
115.    }
116.    @Override
117.    public Medico getMedico_nif(String nif) throws DoesNotExistsException,
    RemoteException {
118.        if (medico_nif.containsKey(nif)) {
119.            return medico_nif.get(nif);
120.        } else {
121.            throw new DoesNotExistsException();
122.        }
123.    }
124.    @Override
125.    public Medico getMedico_cedula(String cedula) throws DoesNotExistsException,
    RemoteException {
126.        if (medico_cedula.containsKey(cedula)) {
127.            return medico_cedula.get(cedula);
128.        } else {
129.            throw new DoesNotExistsException();
130.        }
131.    }
132.    @Override
133.    public Funcionario getFuncionario_name(String name) throws DoesNotExistsException,
    RemoteException {
134.        if (funcionario_name.containsKey(name)){
135.            return funcionario_name.get(name);
136.        } else {
137.            throw new DoesNotExistsException();
138.        }
139.    }
140.    @Override
141.    public Funcionario getFuncionario_numfunc(String numfunc) throws
    DoesNotExistsException, RemoteException {
142.        if (funcionario_numfunc.containsKey(numfunc)) {
143.            return funcionario_numfunc.get(numfunc);
144.        } else {
145.            throw new DoesNotExistsException();
146.        }
147.    }
148.    @Override
149.    public Funcionario getFuncionario_nif(String nif) throws DoesNotExistsException,
    RemoteException {
150.        if (funcionario_nif.containsKey(nif)) {
151.            return funcionario_nif.get(nif);
152.        } else {
153.            throw new DoesNotExistsException();
154.        }
155.    }
156.    @Override
157.    public void AdicionaFunc(String nome, String morada, String nif, String cc,
    LocalDate datanasc, int numfunc) throws RemoteException{
158.
159.        if (funcionarios.containsKey(numfunc)){
160.            System.out.println("Funcionario já existe");
161.        } else {
162.            Funcionario f = new Funcionario(nome, morada, nif, cc, datanasc, numfunc);
163.            AdicionarFunc(f);
164.        }
165.    }
166.   public void AdicionarFunc (Funcionario f){
```

```
167.        this.funcionarios.put(f.getNumfunc(), new Funcionario(f.getNome(),f.getMorada(),
    f.getNif(),f.getCc(),f.getDatanasc(),f.getNumfunc()));
168.    }
169.    @Override
170.    public void AdicionaMed(String nome, String morada, String nif, String cc, LocalDate
     datanasc, String cedula, String especialidade) throws RemoteException{
171.        if (medicos.containsKey(cedula)){
172.            System.out.println("Medico já existe");
173.        } else {
174.            Medico m = new Medico(nome, morada, nif, cc, datanasc, cedula, especialidade
    );
175.            AdicionarMed(m);
176.        }
177.    }
178.
179.    public void AdicionarMed(Medico m){
180.        this.medicos.put(m.getCedula(), new Medico(m.getNome(),m.getMorada(),m.getNif(),
    m.getCc(),m.getDatanasc(),m.getCedula(), m.getEspecialidade()));
181.    }
182.
183.    @Override
184.    public void AdicionaMec(String contador, String dci, String nome, String formafarmac
    eutica, String dosagem, String estadoautorizacao, boolean generico, String titular_aim)
    throws RemoteException{
185.        if (medicamentos.containsKey(contador)){
186.            System.out.println("Medicamento já existe");
187.        } else {
188.            Medicamento medc = new Medicamento(dci, nome, formafarmaceutica, dosagem,
    estadoautorizacao, generico, titular_aim);
189.            AdicionarMedc(medc, contador);
190.        }
191.
192.    }
193.
194.    public void AdicionarMedc(Medicamento medc,String contador){
195.        this.medicamentos.put(contador, new Medicamento(medc.getDci(), medc.getNome(),
    medc.getFormafarmaceutica(), medc.getDosagem(), medc.getEstadoautorizacao(),
    medc.isGenerico(), medc.getTitular_aim()));
196.    }
197.
198.    @Override
199.    public synchronized String LoginUtente (String numutente, String nif) throws
    RemoteException {
200.        if (!this.fichasUtente.containsKey(numutente)) {
201.            System.out.println("Registe-se antes de fazer login.");
202.        } else if (!this.fichasUtente.get(numutente).getUtente().getNif().equals(nif)) {

203.            System.out.println("Password incorreta!");
204.        }
205.        else { return numutente; }
206.        return null;
207.    }
208.
209.    @Override
210.    public synchronized String LoginMedico (String cedula, String nif) throws
    RemoteException {
211.        if (!this.medicos.containsKey(cedula)) {
212.            System.out.println("Registe-se antes de fazer login.");
213.        } else if (!this.medicos.get(cedula).getNif().equals(nif)) {
214.            System.out.println("Password incorreta!");
215.        }
216.        else { return cedula; }
217.        return null;
218.    }
219.
220.
```

```
221.    @Override
222.    public synchronized int LoginFunc (int numfunc, String nif) throws RemoteException {

223.        if (!this.funcionarios.containsKey(numfunc)) {
224.            System.out.println("Registe-se antes de fazer login.");
225.        } else if (! this.funcionarios.get(numfunc).getNif().equals(nif)) {
226.            System.out.println("Password incorreta!");
227.        }
228.        else { return numfunc; }
229.        return 0;
230.    }
231.
232.    public synchronized Utente fornecer_utente(String numutente){
233.        Utente utente = null;
234.        for(String key : fichasUtente.keySet()){
235.            if (key.equals(numutente)){
236.                return fichasUtente.get(key).getUtente();
237.            }
238.        }
239.        return utente;
240.    }
241.
242.    @Override
243.    public void verlistaMedicos() throws RemoteException {
244.
245.        for (Medico umedico : medicos.values()) {
246.            System.out.println(umedico.getNome());
247.            medico_name.put(umedico.getNome(), umedico);
248.            medico_cedula.put(umedico.getCedula(), umedico);
249.            medico_nif.put(umedico.getNif(), umedico);
250.
251.        }
252.    }
253.
254.    @Override
255.    public void verlistaFuncionarios() throws RemoteException {
256.       for (Funcionario umfunc : funcionarios.values()) {
257.            System.out.println(umfunc.getNome());
258.            funcionario_name.put(umfunc.getNome(), umfunc);
259.            funcionario_nif.put(umfunc.getNif(), umfunc);
260.            funcionario_numfunc.put(umfunc.getNumfunc(), umfunc);
261.        }
262.    }
263.
264.    public synchronized Medicao adicionar_medicao(LocalDate data, double peso, int
    altura, double glicemia, int tensaoarterial, int colesterol, int triglicerideos, int
    saturacao, int inr)
265.            throws RemoteException {
266.
267.        Medicao medicao;
268.
269.        medicao = new Medicao(data,peso, altura, glicemia, tensaoarterial,colesterol,
    triglicerideos, saturacao, inr);
270.
271.        return medicao;
272.    }
273.
274.    //MÉTODO CRIADO APENAS PARA O CARREGAMENTO DAS CONSULTAS DO TXT
275.    @Override
276.    public synchronized Consulta adicionar_consulta(Medico med, String numutente, String
     Observacoes) throws RemoteException {
277.        Consulta co = new Consulta(med);
278.        co.addObservacao(Observacoes);
279.
280.        return co;
281.    }
```

```
282.    //MÉTODO CRIADO APENAS PARA O CARREGAMENTO DAS CONSULTAS DO TXT
283.    @Override
284.    public synchronized void criar_prescricoes(Consulta co, String numutente,LocalDate
    data, Medicamento medc, String toma) throws RemoteException {
285.        Prescricao p = new Prescricao(data, medc, toma);
286.        co.addPrescricao(p);
287.        for (String key : fichasUtente.keySet()){
288.            if (key.equals(numutente)){
289.                fichasUtente.get(key).addConsulta(co);
290.            }
291.        }
292.    }
293.
294.    //MÉTODO CRIADO APENAS PARA O CARREGAMENTO DAS CONSULTAS DO TXT
295.    @Override
296.    public synchronized void criar_Observacoes(Consulta co, String observacoes) throws
    RemoteException{
297.
298.        co.addObservacao(observacoes);
299.
300.    }
```