

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**  
**BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO**  
**RELATÓRIO TÉCNICO - PROGRAMAÇÃO ORIENTADA À OBJETOS**

**Arnald Souza**

**Julio Cesar Farias**

**Leonardo Gonçalves Fagote**

**SISTEMA DE GERENCIAMENTO DE HOSPITAL**

**APUCARANA, 2022**

**Arnald Souza**

**Julio Cesar Farias**

**Leonardo Gonçalves Fagote**

## **SISTEMA DE GERENCIAMENTO DE HOSPITAL**

Relatório Técnico do Trabalho Multidisciplinar (TM) apresentado como requisito parcial para obtenção de créditos na disciplina de Programação Orientada à Objeto da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Lucio Agostinho Rocha

**APUCARANA, 2022**

## RESUMO

Neste trabalho, foi desenvolvido um sistema de gerenciamento de hospital. O código foi inteiramente desenvolvido na linguagem de programação Java. O funcionamento do sistema consiste na inserção das informações dos pacientes, médicos e quanto o paciente terá que pagar pela conta, assim como o quanto a administração do hospital tem que pagar para o médico e também a verificação se o paciente ou o médico existe no banco de dados.

**Palavras-chave:** Hospital. Gerenciamento.

## LISTA DE ILUSTRAÇÕES

Figura 1 - Herança da classe Administração pela classe Médico.....	7
Figura 2 - Herança da classe Administração pela classe Paciente.....	7
Figura 3 - Polimorfismo.....	8
Figura 4 - Polimorfismo na classe médico.....	8
Figura 5 - Polimorfismo na classe paciente.....	9
Figura 6 - Design Pattern .....	10
Figura 7 - Tratamento de Exceção .....	11
Figura 8 - Hashmap.....	11
Figura 9 - Janela de Opções .....	12
Figura 10 - Cadastro de Médico.....	13
Figura 11 - Cadastro de Paciente.....	13
Figura 12 - Procura de Médico.....	14
Figura 13 - Procura de Paciente.....	15
Figura 14 - Utilização de Herança.....	16
Figura 15 - Erro capturado pelo Tratamento de Exceção.....	17
Figura 16 - Diagrama de Caso de Uso.....	17
Figura 17 - Diagrama UML de Classe.....	18
Figura 18 - Cadastro Médico.....	19
Figura 19 - Cadastro Paciente.....	19

Figura 20 - Verificar Médico.....	19
Figura 21 - Verificar Paciente .....	20

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>6</b>
<b>2 DESENVOLVIMENTO .....</b>	<b>7</b>
2.1 Utilização de Herança.....	7
2.2 Utilização de Polimorfismo.....	8
2.3 Design Pattern (Padrão de Projetos).....	9
2.4 Tratamento de Exceções.....	10
2.5 Hashmap.....	11
2.6 Link Público do Código Fonte.....	11
<b>3 RESULTADOS .....</b>	<b>12</b>
3.1 Resultados da Utilização de Herança.....	15
3.2 Resultados da Utilização de Polimorfismo.....	16
3.3 Resultados do Tratamento de Exceções.....	16
3.4 Diagramas UML do projeto.....	17
<b>4 CONSIDERAÇÕES FINAIS .....</b>	<b>20</b>
<b>REFERÊNCIAS .....</b>	<b>21</b>

## 1 INTRODUÇÃO

O presente projeto refere-se à aplicação dos assuntos vistos na disciplina Programação Orientada à Objeto, trabalhado na linguagem de programação Java. Foi desenvolvido com o objetivo de simplificar um sistema de gerenciamento de hospital, tornando o sistema simples e funcional.

Vários recursos e conceitos da linguagem Java foram aplicados no código fonte do programa gerado, dentre eles ressaltam-se os assuntos de encapsulamento, herança e polimorfismo.

O corpo do projeto é composto pelos diagramas UML, com os diagramas prontos deu-se início ao desenvolvimento do código, composto pelas classes, classes abstratas, interfaces e seus relacionamentos. Com as mesmas prontas, os métodos foram implementados para cada parte do programa funcionar harmonicamente e de acordo com o que se havia proposto inicialmente.

Com a solução proposta, este projeto tem grande potencial para auxiliar no gerenciamento de um hospital de nível pequeno/intermediário, dessa maneira facilitando o cadastro e atualização de pacientes e médicos.

## 2 DESENVOLVIMENTO

Com a formação da ideia de implementar o sistema para o gerenciamento de hospital, os diagramas UML foram montados. Começando pelo diagrama de caso de uso onde o ator denominado administração faz o intermédio entre médico e paciente por meio do pagamento, cadastro e consulta. Após o diagrama de caso de uso ser feito, foi desenvolvido o diagrama de classe, que constam as seguintes classes: médico, administração, paciente e projeto2.

A seguir, o diagrama de estados é elaborado para a administração, paciente e médico.

O sistema operacional utilizado para a produção do projeto foi o Windows, para a construção dos diagramas UML, utilizou-se o software Umbrello, já o código foi implementado no ambiente de desenvolvimento integrado NetBeans em linguagem Java.

### 2.1 Utilização de Herança

Como a classe Medico e Paciente são subclasses da classe Administracao, isso faz com que os métodos e os atributos sejam compartilhadas para a superclasse. Dessa maneira, a classe Administracao receberá nome e CRM do médico e nome e CPF do paciente.

```
public class Medico extends Administracao {
```

Figura 1 - Herança da classe Administração pela classe Médico

```
public class Paciente extends Administracao {
```

Figura 2 - Herança da classe Administração pela classe Paciente



Por meio da herança, todos os atributos e métodos foram passados para um melhor funcionamento do código, onde a superclasse obtém apenas o necessário.

## 2.2 Utilização de Polimorfismo

No código, o polimorfismo foi usado na classe abstrata administração, de tal forma que o método setName() seja assimilado por outras classes para reutilizá-la.

```

❏ import java.util.ArrayList;

public abstract class Administracao extends JavaApplication21 {

    public abstract void setName(String nome);

}

```

Figura 3 - Polimorfismo

Desse modo, as classes médico e paciente podem aplicar métodos abstratos de Administracao.

```

8      public class Medico extends Administracao {
9
10         String nome;
11         String crm;
12
13
14         public Medico() {}
15
16         public String getNome() {
17             return this.nome;
18         }
19
20         public void setName(String nome) {
21             this.nome = nome;
22         }
23

```

Figura 4 - Polimorfismo na classe médico

```

3
4 public class Paciente extends Administracao {
5     String nome;
6     String cpf;
7
8     public Paciente() {}
9
10    public String getNome() {
11        return this.nome;
12    }
13
14    public void setNome(String nome) {
15        this.nome = nome;
16    }

```

Figura 5 - Polimorfismo na classe paciente

Com isso, com a implementação do polimorfismo, não foram necessários criar métodos novos para pegar os dados, apenas replicar os já existentes.

### 2.3 Design Pattern (Padrão de Projetos)

O Design Pattern escolhido foi o Iterator, esse padrão de projeto foi escolhido, porque ele define uma coleção de objetos. Tal projeto será aplicado na classe médico, pois ele mostrará um a um os médicos que já estão cadastrados na administração. Dessa maneira, a administração poderá visualizar quais médicos estão cadastrados no sistema.

```

private ArrayList<Medico> lista;

public interface Iterator{
    public abstract boolean temProximo();
    public abstract Object proximo();
}

public class ItemIterator implements Iterator{

    private ArrayList<Medico> lista;
    int pos = 0;

    public ItemIterator(ArrayList<Medico> lista){
        this.lista = lista;
    }

    public Object proximo() {
        Object item = lista.get(index:pos);
        pos++;
    }
}

```

Figura 6 - Design Pattern

## 2.4 Tratamento de Exceções

O tratamento de exceção está sendo usado para controlar o tamanho do CRM de médico, ou seja, se o CRM for maior que seis o programa vai printar uma mensagem de erro e não irá cadastrar o médico.

```
//-----
// TRATAMENTO DE EXCEÇÃO PARA CRM
try{
    if(cod.length() != 6){
        lista.remove(new Medico( nome:m1.getNome(), crm:m1.getCRM()));
        throw new Exception( message: "CRM invalido, o medico nao foi cadastrado!!");
    }
    lista.add(new Medico( nome:m1.getNome(), crm:m1.getCRM()));
} catch(Exception e){
    System.out.println( x:e.getMessage());
}
//-----
```

Figura 7 - Tratamento de Exceções

## 2.5 Hashmap

O HashMap trabalha com o conceito de key-value pairs, ou seja, cada elemento de sua lista possui uma chave e valor associado, assim podemos realizar uma busca rápida pela chave que desejamos, sem precisar percorrer toda lista ou saber o index/posição que desejamos consultar.

As chaves usadas neste presente projeto foram nome e crm para o médico, e nome e cpf para o paciente, como mostra a figura a seguir:

```
static HashMap<String, String> MedicosArray = new HashMap<>();
static HashMap<String, String> PacientesArray = new HashMap<>()

public static void cadastrar(Medico medico) {
    MedicosArray.put( key:medico.nome, value:medico.crm);
}

public static void cadastrarP(Paciente paciente) {
    PacientesArray.put( key:paciente.nome, value:paciente.cpf);
}
```

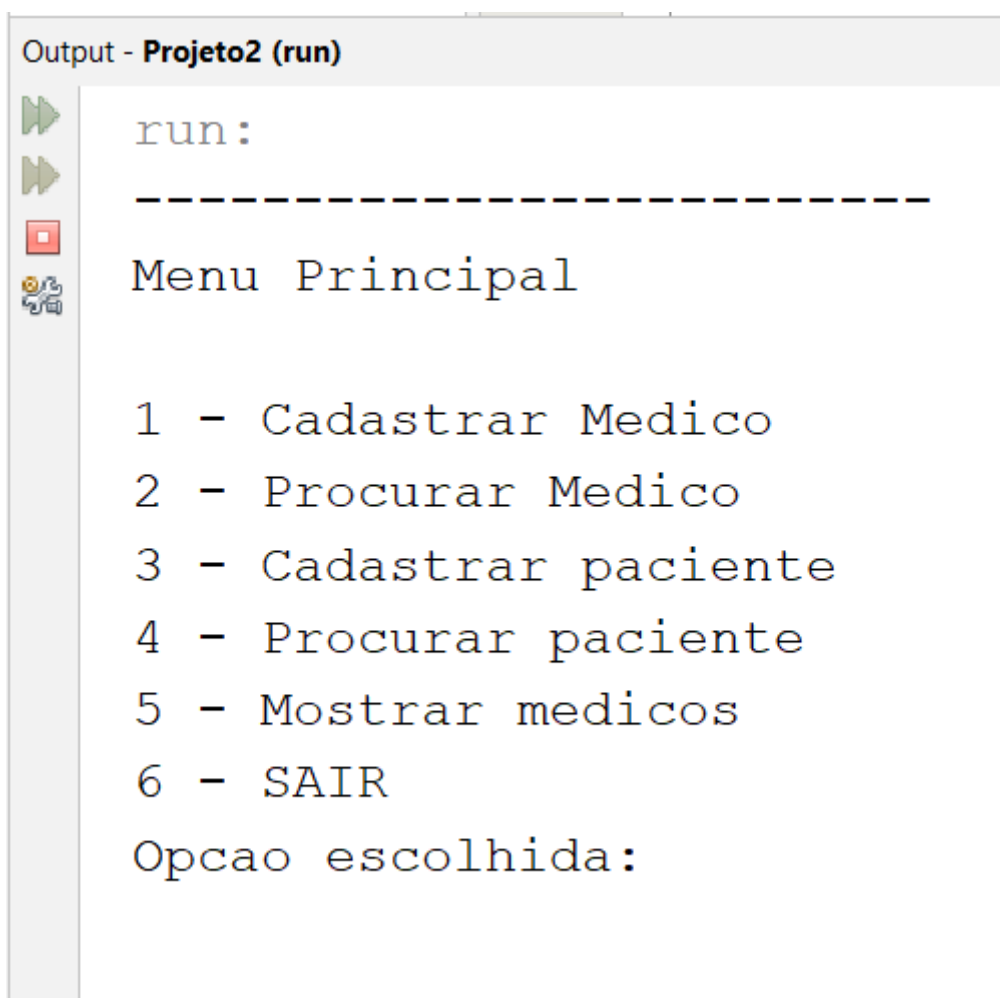
Figura 8 - Hashmap

## 2.6 Link Público do Código-Fonte

<https://github.com/fagote/Trabalho-POO>

### 3 RESULTADOS

A execução do código se dá pelo seguinte processo, ao executar o programa a partir do arquivo “Projeto2.java” a função principal inicializará fazendo que apareça um menu de opções na saída. Na saída o usuário vai escolher entre 6 opções cadastrar médico ou paciente, consultar médico ou paciente, exibir médico e sair do programa.

The image shows a screenshot of a Java IDE's output window. The window title is "Output - Projeto2 (run)". On the left side of the window, there are four icons: a green play button, a green right-pointing arrow, a red square, and a bug icon. The output text is as follows:

```
run :  
-----  
Menu Principal  
  
1 - Cadastrar Medico  
2 - Procurar Medico  
3 - Cadastrar paciente  
4 - Procurar paciente  
5 - Mostrar medicos  
6 - SAIR  
Opcao escolhida:
```

Figura 9 - Janela de opções

Se a opção escolhida for a de cadastrar médico, o nome e o CRM serão armazenados no banco de dados, se a opção selecionada for cadastrar paciente o nome e o CPF serão também armazenados no banco de dados.

## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 1

Nome: Jurema

CRM: 12345PR

---

Figura 10 - Cadastro de Médico

## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 3

Nome: jurebaldo

CPF: 13245678900

Figura 11 - Cadastro de Paciente

Se a opção escolhida for a de procurar o médico a partir do nome será buscado tal objeto para então mostrar que ele existe ou não e apresentar o seu CRM armazenados. Se a opção escolhida for a de procurar o paciente a partir do nome será buscado tal objeto para então mostrar que ele existe ou não e apresentar o seu CPF armazenados.

## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 2

{Jurema=12345PR}

O que deseja buscar?

Jurema

Valor da Chave Jurema = 12345PR

Figura 12 - Procura de Médico

## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 4

```
{jurebaldo=13245678900}
```

O que deseja buscar?

jurebaldo

Valor da Chave jurebaldo = 13245678900

Figura 13 -Procura de Paciente

### 3.1 Resultados da Utilização de Herança

A herança foi usada para que a classe Administração recebesse todos os métodos e atributos para a saída de dados e assim armazená-los nessa classe para poder então consultá-los.



## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 4

```
{jurebaldo=13245678900}
```

O que deseja buscar?

jurebaldo

Valor da Chave jurebaldo = 13245678900

Figura 14 - Utilização de Herança

### 3.2 Resultados da Utilização de Polimorfismo

Uma das necessidades para a construção de uma administração, foi a de diferenciar o que seria o nome do médico e o nome do paciente. A solução para isso foi a utilização de polimorfismo onde a partir da mesma instância tivessem acesso às classes paciente e médico.

### 3.3 Resultados do Tratamento de Exceções

O tratamento de exceção foi usado com objetivo de mostrar um aviso caso os caracteres CRM sejam ultrapassados em seis. A partir disso o programa dispara uma mensagem “CRM invalido, o medico nao foi cadastrado! !” e não cadastrará o médico.

## Menu Principal

- 1 - Cadastrar Medico
- 2 - Procurar Medico
- 3 - Cadastrar paciente
- 4 - Procurar paciente
- 5 - Mostrar medicos
- 6 - SAIR

Opcao escolhida: 1

Nome: Jurema

CRM: 12345PR

CRM invalido, o medico nao foi cadastrado!!

Figura 15 - Erro capturado pelo Tratamento de Exceções

### 3.4 Diagramas UML do projeto

Diagrama UML de Casos de Uso:

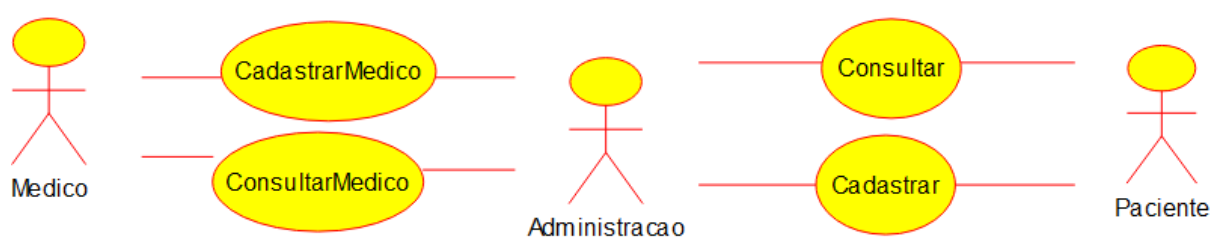


Figura 16 - Diagrama UML de Casos de Uso

## Diagrama UML de Classes:

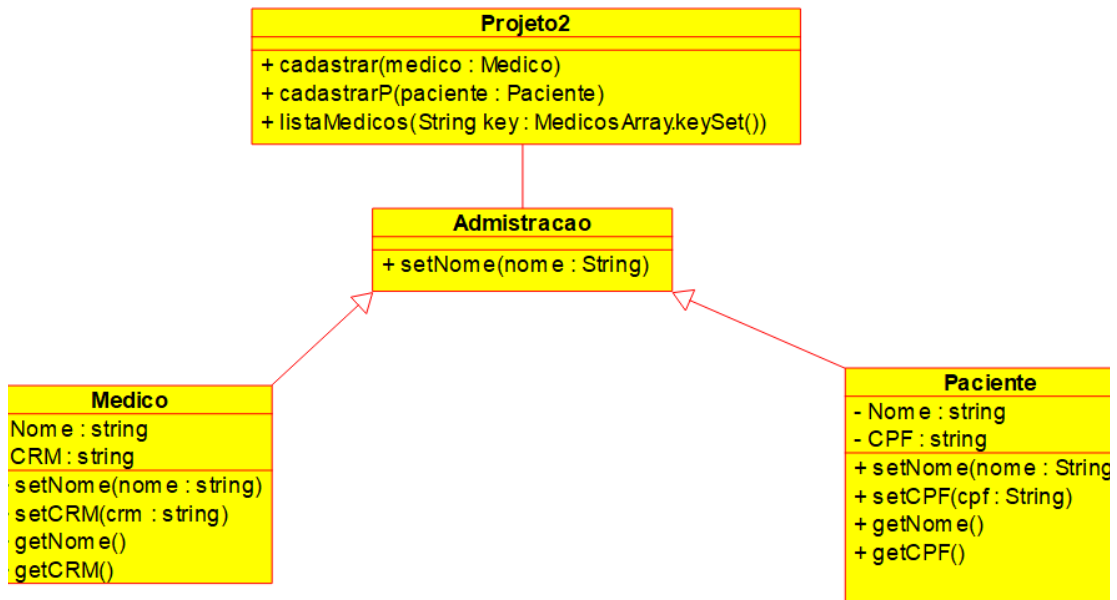


Figura 17 - Diagrama UML de Classes

## Diagramas UML de Sequência:

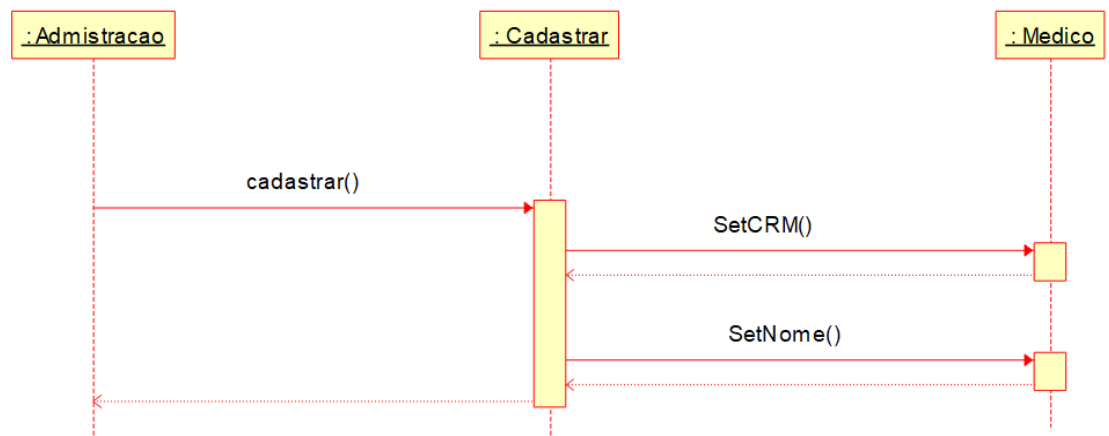


Figura 18 - Diagrama de Sequência Cadastro Médico

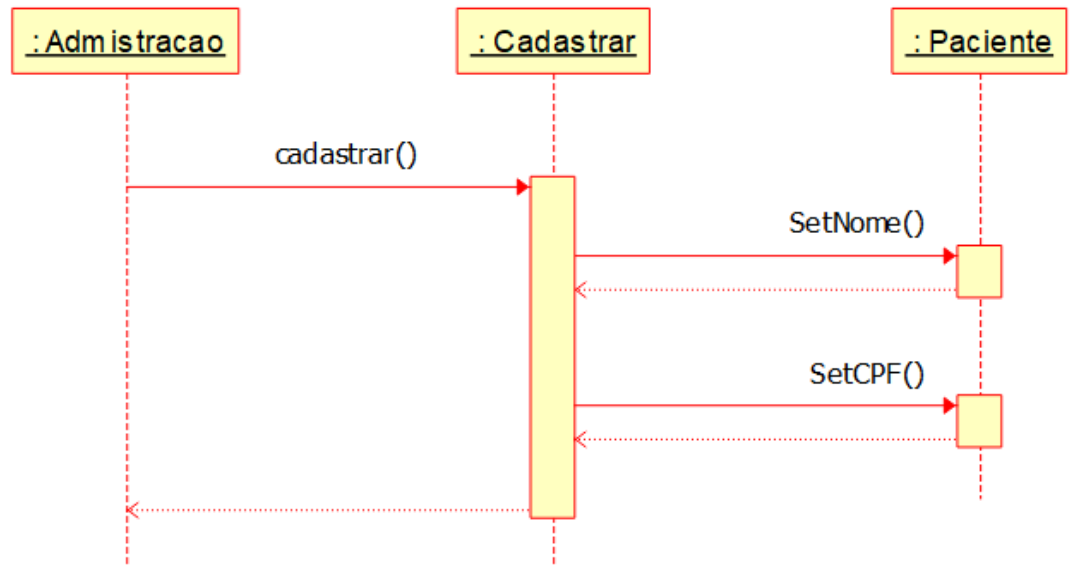


Figura 19 - Diagrama de Sequência Cadastro Paciente

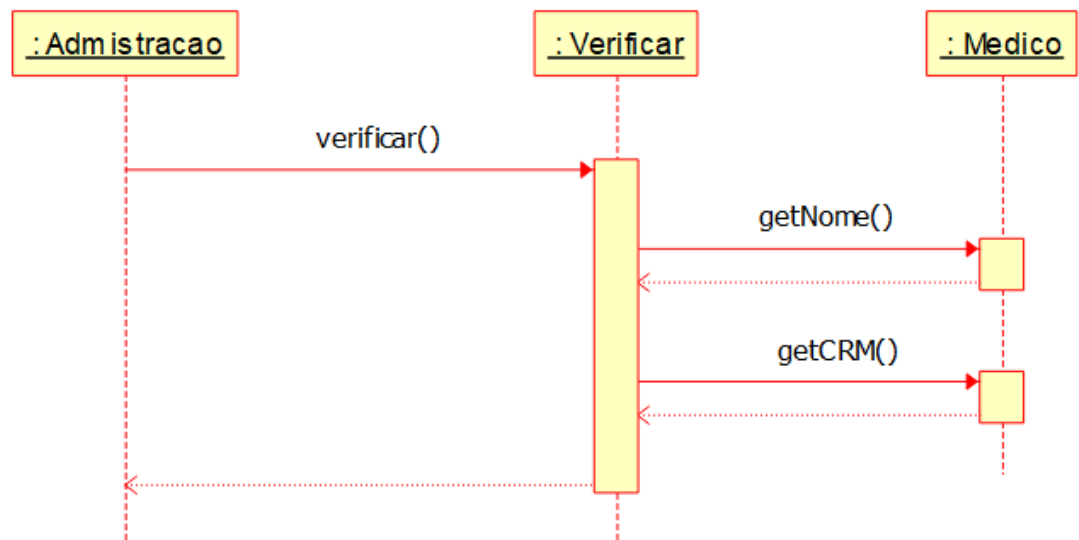


Figura 20 - Diagrama de Sequência Verificar Médico

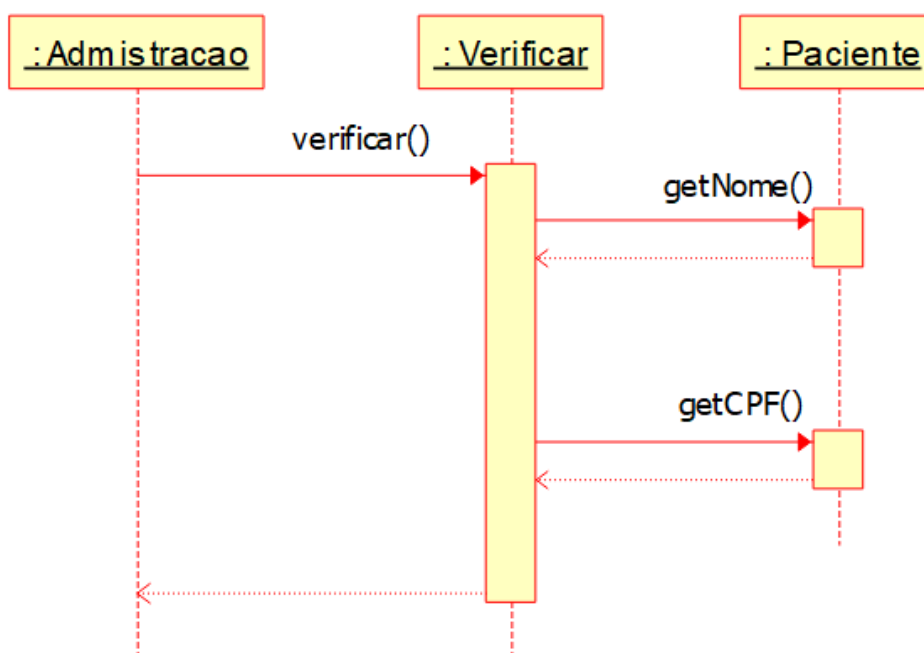


Figura 21 - Diagrama de Sequência Verificar Paciente

#### 4 CONSIDERAÇÕES FINAIS

Levando em conta o que foi observado e descrito, alcançamos resultados satisfatórios com a implementação do código, pois atendemos aos critérios propostos na atividade, além de desenvolver um programa que auxilia hospitais de nível pequeno/intermediário a realizar cadastro de médicos e pacientes, também realizando a busca de um médico ou paciente específico.

Vale ressaltar que o projeto consolidou o aprendizado de diversas técnicas de programação em java, que foram estudadas em aula, como herança, tratamento de exceção, polimorfismo e construção de diagramas UML.

## REFERÊNCIAS

DEITEL, Paul; DEITEL, Harvey. **Java como Programar**. 10. ed. São Paulo: Pearson, 2017.

SIERRA, Kathy; BATES, Bert. **Use a cabeça!** Java. 2. ed. Rio de Janeiro, RJ: Alta Books, 2007.

ZIVIANI, Nivio. **Projeto de algoritmos:** com implementações em Java e C++. São Paulo, SP: Cengage Learning, 2007.