



# LICENCIATURA ENGENHARIA INFORMÁTICA E MULTIMÉDIA

## FUNDAMENTOS DE SISTEMAS OPERATIVOS

---

### Trabalho Prático 1

---

DATA: 12 DE NOVEMBRO DE 2020

*Docentes:*

Eng. Carlos Gonçalves

Eng. Jorge Pais

Eng. Carlos Carvalho

*Realizado por:*

Luís Fonseca - 45125

João Rodrigues - 45145

*Grupo: 4*

# Índice de Conteúdos

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>O que é um Processo?</b>	<b>4</b>
<b>3</b>	<b>Comunicação entre processos em Java</b>	<b>4</b>
<b>4</b>	<b>A Mensagem</b>	<b>4</b>
<b>5</b>	<b>Diagrama de Atividades da Aplicação</b>	<b>5</b>
<b>6</b>	<b>Diagrama Swimplane da Aplicação</b>	<b>6</b>
<b>7</b>	<b>GUI do Chat</b>	<b>7</b>
<b>8</b>	<b>Canal de Comunicação</b>	<b>8</b>
8.1	Interface Do Canal De Comunicação . . . . .	8
8.2	Como Criar um Canal de Comunicação? . . . . .	8
8.3	Como escrever no Canal de Comunicação . . . . .	9
8.4	Como ler no Canal de Comunicação . . . . .	10
8.5	Como fechar o Canal de Comunicação . . . . .	11
8.6	Outros métodos desta classe . . . . .	11
<b>9</b>	<b>Máquina de Estados</b>	<b>12</b>
<b>10</b>	<b>Diagrama de Classes</b>	<b>16</b>
<b>11</b>	<b>Resultados</b>	<b>17</b>
<b>12</b>	<b>Conclusões</b>	<b>19</b>
<b>13</b>	<b>Anexo</b>	<b>20</b>
13.1	ClasseGui Chat . . . . .	20
13.2	Mensagem . . . . .	27
13.3	ICanalComunicaao . . . . .	28
13.4	CanalComunicacao . . . . .	29
13.5	MaquinaEstados . . . . .	33
<b>14</b>	<b>Bibliografia</b>	<b>36</b>

**List of Figures**

1	Diagrama de Atividades do Primeiro Trabalho Prático . . . . .	5
2	Diagrama Swimlane do Primeiro Trabalho Prático . . . . .	6
3	GUI do chat . . . . .	7
4	Diagrama de Classes do Primeiro Trabalho Prático . . . . .	16
5	Processo 1 a comunicar com o mesmo ficheiro que o Processo2 . . . .	17
6	Processo 2 a comunicar com o mesmo ficheiro que o Processo1 . . . .	18

# 1 Introdução

Neste relatório vamos abordar como foi realizado o primeiro trabalho prático da disciplina de Fundamentos de Sistemas Operativos. Vamos discutir o problema que teria de ser resolvido, as tecnologias usadas e finalmente iremos justificar, conforme necessário, a solução apresentada.

Este primeiro trabalho tem como objetivos principais a aprendizagem de programação em multi-processo e a comunicação entre processos usando um canal de comunicação. Assim sendo foi-nos proposto a realização de um Chat que vai ser aberto mais que uma vez, de cada vez que é aberto um novo Chat, é criado um novo processo.



## 2 O que é um Processo?

*”Em computação, um **processo** é uma instância de um programa de computador que está sendo executada”*

Ou seja, ao criarmos e correremos um programa, criamos um processo, mesmo que este processo apenas dure milésimas de segundo, o processo teve de ser inicializado, aguardar a sua vez para aceder aos recursos da máquina, ter sido **executada** e terminar.

Em Java, um processo pode ser criado utilizando a classe `ProcessBuilder`. Um processo é uma instância da classe `ProcessBuilder` tendo como parâmetro de entrada o “nomeProcesso” do tipo `String`. No parâmetro “nomeProcesso” consta o nome do processo executável mais o respetivo caminho (path). De maneira prática, todos os ficheiros JAVA, com um método static ***void main (String[] args)***, pode iniciar um processo.

## 3 Comunicação entre processos em Java

Em JAVA, a comunicação entre processos pode ser implementada de diversas formas, neste trabalho foi criado um buffer(buffer esse que apenas contém uma mensagem) onde seriam lidas e escritas as mensagens a serem transmitidas. Assim foi implementada uma comunicação entre processos através de memória partilhada. De modo a permitir e a facilitar a comunicação entre processos neste trabalho, foram criadas duas classes, a classe `CanalComunicação` e a classe `Mensagem`.

## 4 A Mensagem

A classe `Mensagem` é uma classe que no seu construtor pode receber três argumentos sendo eles o **id** o **tipo** e a **mensagem**, ou caso apenas seja preciso inicializar a classe podemos usar o construtor que não recebe nenhum argumento, podendo depois introduzir-se os valores de id, tipo e mensagem usando os getters criados.

Atributos usados na classe `Mensagem` e o que fazem:

- O id indica quantas mensagens foram já enviadas **por um certo indivíduo.**
- O tipo possibilita a escolha da pessoa que está a enviar se quer enviar a mensagem para todos os “indivíduos” ou apenas para um.
- A mensagem, é a mensagem que o indivíduo que escreve na `TextArea` envia.

Caso o utilizador queira inserir os valores de id, tipo e mensagem, não no construtor mas sim mais tarde ao longo do código, pode alterar estes valores usando os Getters e Setters criados.

## 5 Diagrama de Atividades da Aplicação

Antes de passar para a implementação, em código, da aplicação realizada, procedemos à realização de um diagrama de atividades, contendo todos os passos para o correto funcionamento.

Como primeiro passo, iremos ter como estado inicial, o utilizador iniciar a aplicação, abrindo uma GUI com vários comandos(essa GUI pode ser encontrada no tópico a seguir).

De seguida verificamos se o utilizador iniciou o canal de comunicação, através da condição que está no diagrama. Caso tenha selecionado, mostra uma janela onde necessita de seleccionar um ficheiro(com a extensão, .dat, .txt, etc). Antes de passar para o estado de escrever e ler mensagens, foi verificado se o ficheiro escolhido era o correto. Caso seja esse o ficheiro, então irá passar para o estado seguinte, que consiste em escrever uma mensagem para o buffer. A partir daqui, sempre que dois processos forem iniciados, eles podem receber e enviar mensagens, desde que partilhem o mesmo ficheiro no canal de comunicação

Finalmente, passamos aos dois últimos estados, o ler e escrever mensagem. Sempre que é escrita uma mensagem dentro do buffer, o outro processo precisa de esperar um 1 segundo para ler a mensagem que lhe foi enviada.

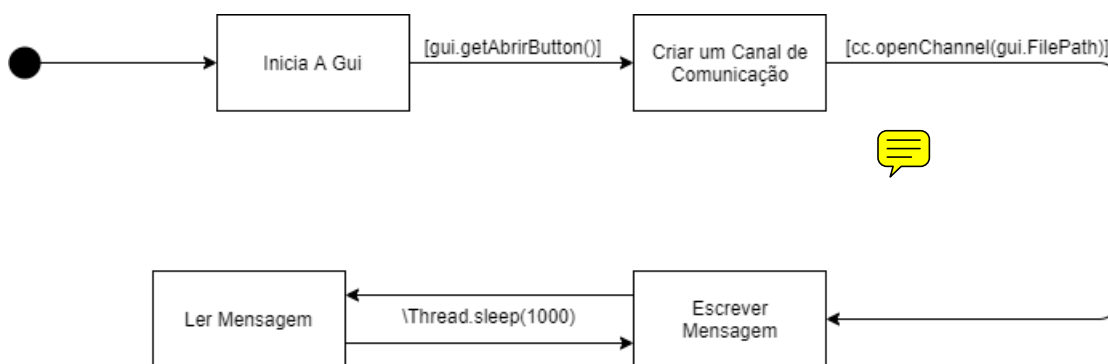


Figura 7.1 Diagrama de Atividades do Primeiro Trabalho Prático

## 6 Diagrama Swimlane da Aplicação

De seguida foi pedido a realização de um diagrama Swimlane. Um diagrama Swimlane é As swimlanes servem para representar a sincronização e a comunicação de objetos entre tarefas diferentes (neste caso estamos a falar em processos diferentes comunicarem entre si).

A realização deste diagrama não ficou muito distante do diagrama acima construído, é de reparar que os estados de ler e escrever iram funcionar em processos diferentes, mas comunicando no mesmo canal de comunicação.

É de salientar o último estado, que consiste em terminar a aplicação, terminando a ligação ao canal de comunicação.

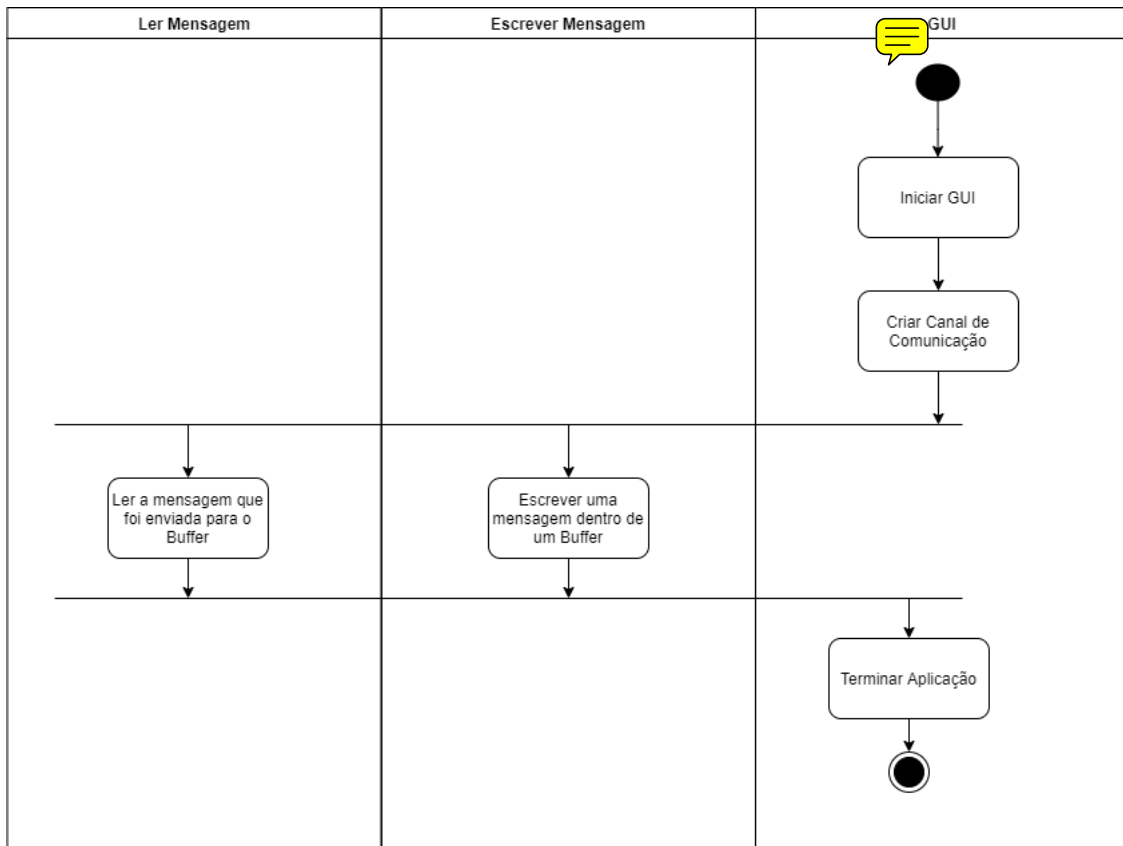


Figure 2: Diagrama Swimlane do Primeiro Trabalho Prático

## 7 GUI do Chat

A GUI do chat é a interface gráfica que possibilita ao utilizador da aplicação criada, interagir com a mesma. A GUI conta com um botão de Browse que permite ao utilizador escolher o ficheiro que será o canal de comunicação, conta também com um botão abrir que vai inicializar o canal de comunicação, tem 4 botões rádio que permite ao utilizador escolher qual o tipo de mensagem que quer enviar, tem 2 áreas de texto, uma que é onde o utilizador escreve a mensagem que deseja enviar e outra que aparece as mensagens enviadas, e para finalizar tem 2 botões que são o terminar que fecha o canal de comunicação e apaga-o e o botão enviar que envia a mensagem escrita pelo utilizador.

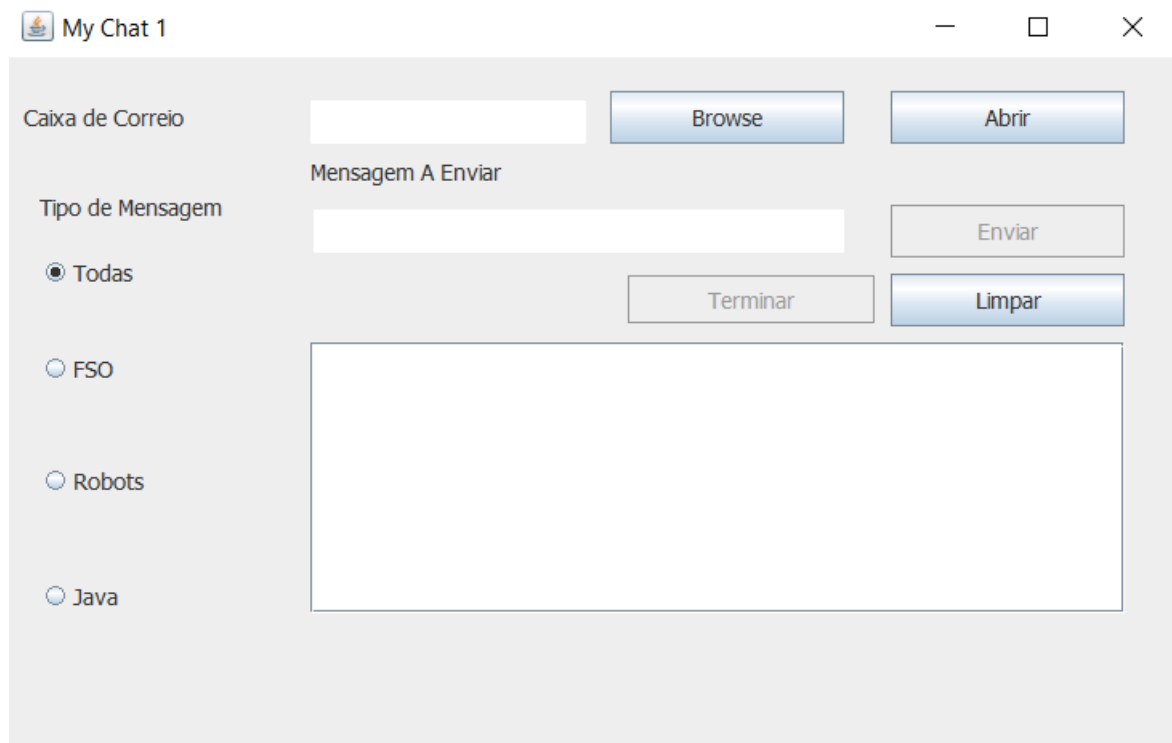


Figure 3: GUI do chat



## 8 Canal de Comunicação

A classe `CanalComunicacao` é uma classe responsável por facilitar a comunicação entre processos. Muito resumidamente o que esta classe faz é alocar um espaço em memória onde serão armazenadas as mensagens que **irão** ser escritas. De modo a alocar o espaço em memória necessário, foi utilizada a classe `MappedByteBuffer`, esta classe nativa do JAVA, sendo capaz de guardar e ler para a memória do PC.

### 8.1 Interface Do Canal De Comunicação

Antes de passar para a implementação do canal de comunicação, foi criada uma interface que engloba todos os métodos necessários para o correto funcionamento desta classe. Os métodos criados para esta interface foram os seguintes:

- `openChannel(String Filename)`: cria um canal de comunicação, passando qualquer tipo de ficheiro, com uma dada extensão;
- `write(Mensagem m)`: permite escrever uma mensagem dentro do canal de comunicação;
- `receberMensagem()`: permite receber/ler uma mensagem do buffer;
- `closeChannel()`: permite terminar a ligação do canal da comunicação e a leitura e escrita do ficheiro;

De seguida, podemos ver a interface criada para esta classe:

---

```
public interface ICanalComunicacao {  
    public boolean openChannel(String Filename) throws IOException;  
    public void write(Mensagem m) throws IOException;  
    public String receberMensagem();  
    public void closeChannel();  
}
```

---

**Listing 1: Interface do Canal de Comunicação**

Nos tópicos a seguir, iremos explicar o que está por detrás de cada método criado com mais detalhe.

### 8.2 Como Criar um Canal de Comunicação?

O primeiro passo para a criação do nosso canal de comunicação foi a criação do ficheiro onde seriam lidos e escritos bytes. para isso criamos o método `openChannel()`, que serve para criar um canal de comunicação, passando um ficheiro com

a extensão ".dat". De seguida, procedemos à alocação do ficheiro para memória, através da classe `RandomAccessFile()` que recebe como argumentos o ficheiro e uma string que indica se o ficheiro é apenas para leitura, ou para escrita ou ambos. Finalmente criamos o buffer propriamente dito, mapeando o ficheiro carregado em memória. Exemplificado através excerto de código que se segue.

```
@Override
public boolean openChannel(String filename) throws IOException{
    ficheiro = new File(filename);
    try {
        this.raf = new RandomAccessFile(this.ficheiro, "rw");
        this.canal = this.raf.getChannel();
        System.out.println("CANAL INICIAL: "+this.canal);
    } catch (FileNotFoundException e) {
        return false;
    }try {
        buffer =
            canal.map(FileChannel.MapMode.READ_WRITE,0,MAX_DIM_BUFFER);
        System.out.println("BUFFER INICIAL: "+buffer);
    } catch (IOException e) {
        return false;
    }
    return true;
}
```

Listing 2: Criação do Canal de Comunicação

### 8.3 Como escrever no Canal de Comunicação

Na nossa implementação, de modo a escrever usamos o método `write(MyMensagem m)`, sendo `m` a mensagem a ser escrita.

O método `write(Mensagem m)` permite escrever uma mensagem no buffer. Era possível este método usar apenas os métodos disponíveis na classe `mappedByteBuffer`, no entanto tal poderia gerar erros caso dois processos quisessem ler e escrever simultaneamente, assim precisamos de bloquear o ficheiro e apenas permitir a escrita à classe que tentou aceder primeiro o ficheiro, pondo a outra em espera. Para isso foi usado a classe `FileLock`, esta classe, permite bloquear uma região no ficheiro ou arquivo a ser lido, garantindo a integridade das mensagens.

Tendo garantido a integridade da mensagem a ser escrita podemos então começar a escrever. O primeiro passo será colocar a posição do buffer onde queremos escrever, para isso usamos o método da classe `mappedByteBuffer` `buffer.position(0)`, e

de seguida colocamos o id e o tipo da mensagem, e para isso, usamos os métodos `buffer.putInt()`, para o ID é necessário incrementar sempre que é enviada uma mensagem, quando o utilizador selecionar o botão "enviar", por isso sempre que escrevemos o ID da mensagem dentro do buffer, incrementamos de um em um o valor do ID dessa mensagem a ser enviada, através da variável `mensagemID`. Finalmente removemos o lock no ficheiro para permitir o acesso por outras classes. Através do código em baixo exemplificamos uma possível implementação.

---

```
@Override
public void write(Mensagem m) throws IOException {
    System.out.println("CANAL ENVIAR: "+this.canal);
    fl = canal.lock(0,canal.size(),false);
    char c;
    buffer.position(0);
    buffer.putInt(m.getId());
    m.setId(m.getId()+1);
    setMensagemID(m.getId());
    buffer.putInt(m.getTipo());
    for (int i= 0 ; i< m.getTexto().length(); ++i){
        c= m.getTexto().charAt(i);
        buffer.putChar(c);
    }
    buffer.putChar('\0');
    fl.release();
}
```

---

Listing 3: Método que permite escrever mensagens no Buffer

## 8.4 Como ler no Canal de Comunicação

O método `receberMensagem()` permite ler uma mensagem no buffer, a realização deste método, permite retornar a posição do buffer, onde foi colocada a mensagem do utilizador, e no final, de seguida usamos o método `getChar()`, permitindo retornar o conjunto de chars que tenha sido colocado dentro do buffer, no final é retornada a string da nossa mensagem.

---

```
@Override
public String receberMensagem() throws IOException {
    String msg = new String();
    try {
        fl = canal.lock(0,canal.size(),false);
        char c;
        this.buffer.position(8);
```

---

```
        while ((c=buffer.getChar())!='\0') {  
            msg += c;  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        fl.release();  
    }  
    return msg;  
}
```

---

Listing 4: Método que permite ler mensagens no Buffer

## 8.5 Como fechar o Canal de Comunicação

Por último era necessário que, fosse feita a opção de caso o utilizador quisesse terminar a aplicação. Para isso foi criado o método *closeChannel()*. O que este método faz é terminar a ligação ao canal de comunicação, e limpar o buffer. O limpar aqui surgiu de ideia para quando o utilizador tenha interesse em voltar a iniciar a aplicação, não ficando com as mensagens anteriores que tinha enviado antes de fechar a aplicação.

---

```
@Override  
public void closeChannel() {  
    try {  
        limpar();  
        canal.close();  
        raf.close();  
    } catch (IOException e) {  
        canal = null;  
        raf = null;  
    }  
}
```

---

Listing 5: Método que fecha a ligação do canal de comunicação

## 8.6 Outros métodos desta classe

O método *limpar()* permite limpar o buffer, reiniciando a partir da posição zero do buffer, este método pode ser usado para limpar um existente.

No final são criados *get's* e *set's* para as variáveis que correspondem ao retorno do *id* e do tipo da mensagem.

## 9 Máquina de Estados

Como último passo deste projeto, criamos uma classe de nome "MáquinaEstados", é nesta classe onde é possível escrever as mensagens dentro do canal de comunicação, e as ler as mesmas.

Para esta máquina de estados, opta-mos por criar um autómato não bloqueante. A sua característica principal tem a ver com tempo de execução, ou seja, é finito, mínimo e limitado à execução das ações de uma atividade.

A execução em código desta máquina de estados foi baseada no diagrama de atividades, em cima.

Como estado inicial, verificamos se o botão de "Abrir" na GUI foi selecionado. Caso seja "True", então transita para o próximo estado, onde é possível ser criado um canal de comunicação.

Neste estado, verificamos se o ficheiro escolhido, é o correto, e caso o ficheiro escolhido é o correto, podemos transitar para o próximo estado


Os próximos dois estados, consistem em escrever e ler mensagens dum respetivo buffer. O estado *WRITE\_MESSAGE* consiste em escrever uma mensagem para dentro de um buffer. Neste caso, o que é escrito é o ID da mensagem, o tipo de processo selecionado e o texto da mensagem.




O estado a seguir corresponde ao *READ\_MESSAGE*, neste estado é onde podemos ler uma mensagem, para isso lê-mos o id, o tipo e o texto da mensagem. Ao ser lido, é apresentado na caixa de texto da GUI, a mensagem com o ID, o tipo de processo selecionado e o texto escrito dentro do buffer

É de salientar também nestes dois estados verificamos os diferentes tipos de processos. Ou seja, caso tenhamos iniciado dois processos, ao selecionar-mos um processo do tipo "FSO" e outro do tipo "Java", nenhum dos processos recebe mensagens, visto que são processos de diferentes tipos. Outro ponto a salientar dos tipos é que, quando um processo tem o tipo selecionado de "Todas", todos os processos podem receber essas mensagens, qualquer que seja o tipo de processo que sejam

No final, criamos um último estado de nome *TERMINATE* que consiste em terminar a ligação ao canal de comunicação e limpar o buffer, não ficando mensagens dentro do buffer, quando fechar a aplicação.

---


  
@Override

```
public void run() {  
    while(true) {  
        switch(numState) {  
  
            case INITIAL_STATE:   
                System.out.println("ESTOU NO INICIAL");  
                if(gui.getAbrirButton())  
                    numState = CREATE_CHANNEL;  
                break;  
  
            case CREATE_CHANNEL:  
                System.out.println("ESTOU NO CREATE");  
                try {  
                    if(cc.openChannel(gui.FilePath))  
                        numState = READ_MESSAGE;  
  
                } catch (IOException e2) {  
                    e2.printStackTrace();   
                }  
                break;  
  
            case WRITE_MESSAGE:  
                System.out.println("ENTREI NO WRITE");  
  
                if(gui.isFSORadio()) {  
                    tipoGeral = 1;  
                } else if(gui.isRobotsRadio()) {  
                    tipoGeral = 2;  
                } else if(gui.isJavaRadio()) {  
                    tipoGeral = 3;  
                } else {  
                    tipoGeral = 0;  
                }  
                String s;  
                s = gui.getEnviarTextArea().getText();  
                System.out.println("MENSAGEM ANTERIOR: "+cc.receberID());   
                mensagem.setId(cc.receberID()+1);  
                mensagem.setTipo(tipoGeral);  
                mensagem.setTexto(s);
```

```
try {
    cc.write(mensagem);
} catch (IOException e) {
    e.printStackTrace();
}
numState = READ_MESSAGE;
break;

case READ_MESSAGE:
    try {
        Thread.sleep(1000);

        int id = cc.receberID();
        int tipo = cc.receberTipo();
        String texto = cc.receberMensagem();

        if(gui.isTodasRadio()) { 
            gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo:
            "+tipo+" Mensagem: "+texto+ "\n");

        }else if(gui.isFSORadio() && (tipo == 1 || tipo ==0)) {

            gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo:
            "+tipo+" Mensagem: "+texto+ "\n");

        }else if(gui.isRobotsRadio() && (tipo == 2 || tipo ==0)) {

            gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo:
            "+tipo+" Mensagem: "+texto+ "\n");

        }else if(gui.isJavaRadio() && (tipo == 3 || tipo ==0)) {

            gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo:
            "+tipo+" Mensagem: "+texto+ "\n");

        }else {
            gui.setMensagensRecebidasTextArea("No est a receber
            nenhuma mensagem \n");
        }
    } catch (InterruptedException e1) {e1.printStackTrace();}
    break;
```

```
        case TERMINATE_APP:
            System.out.println("TERMINATE");
            cc.closeChannel();
            break;
    }
}
```

---

Listing 6: Máquina de Estados da aplicação em desenvolvimento



## 10 Diagrama de Classes

Foi elaborado um diagrama de classes, para a representação de todas as classes, e os seus respetivos métodos criados. Também é mostrado os diferentes atributos que foram usados para aceder às outras classes, permitindo de maneira mais fácil aceder às outras classes, de maneira facilitar a implementação dos diferentes métodos criados.

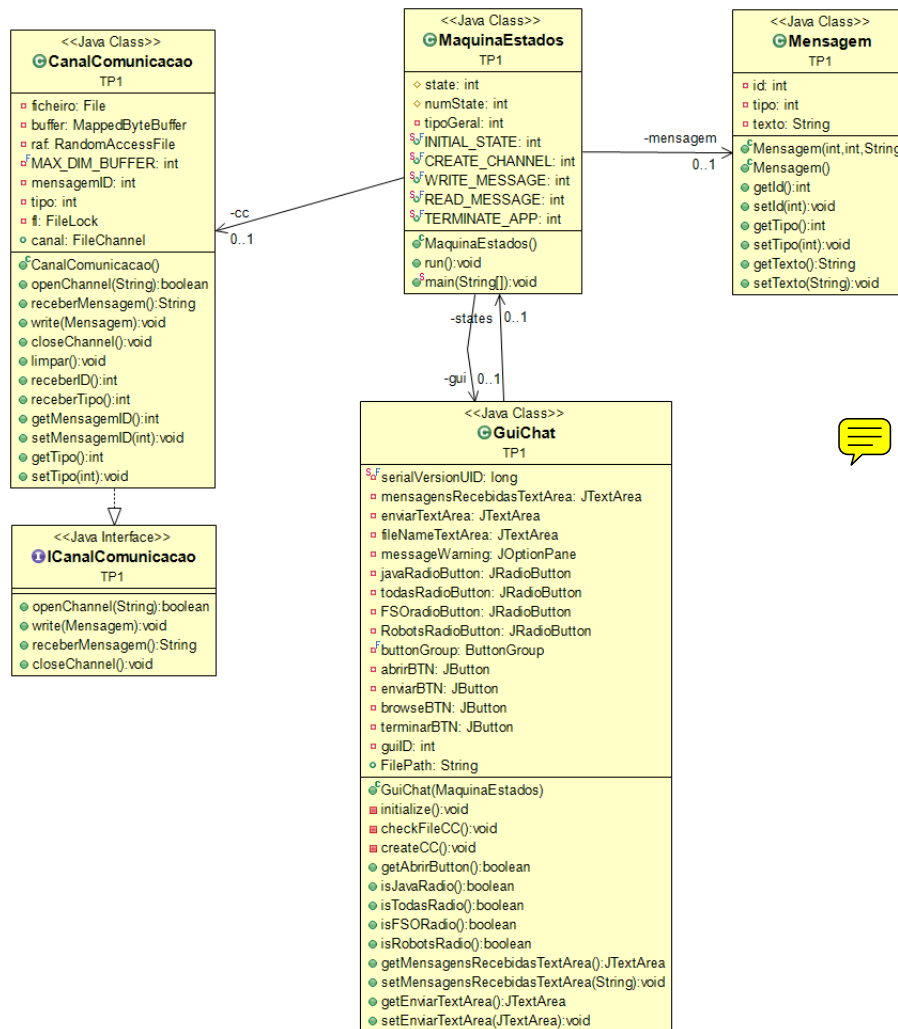


Figure 4: Diagrama de Classes do Primeiro Trabalho Prático

## 11 Resultados

Para comprovar o correto funcionamento da aplicação, iniciamos dois processos, enviando mensagens entre si, neste caso selecionamos um do tipo "Todas" e outro do tipo "Robots", já com o ficheiro selecionado e com o canal de comunicação aberto.

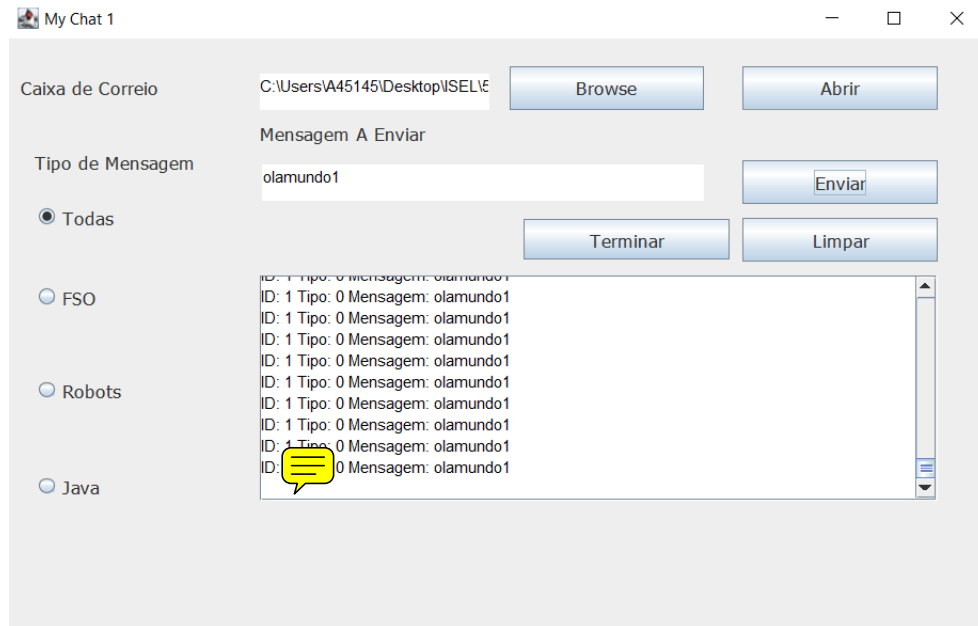


Figure 5: Processo 1 a comunicar com o mesmo ficheiro que o Processo2

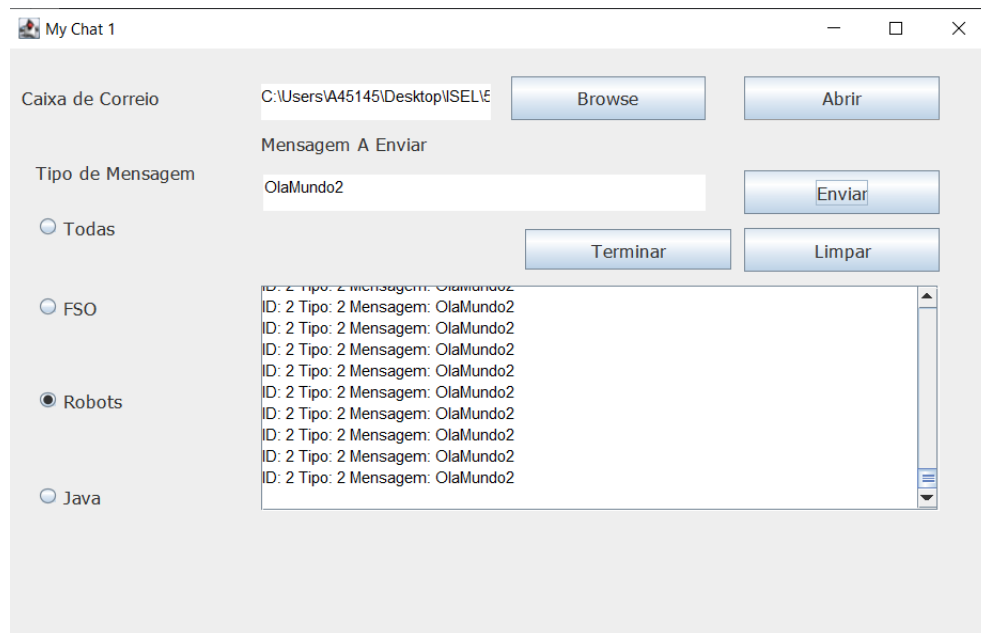


Figure 6: Processo 2 a comunicar com o mesmo ficheiro que o Processo1

Como podemos verificar pelas imagens, o processo 1 enviou uma mensagem para o processo 2 com sucesso, incrementando o ID sempre que envia uma mensagem nova, verifica o tipo de processo seleccionada, e a mensagem que foi escrita pelo processo 1, provando o correto funcionamento da aplicação.

## 12 Conclusões

Com a realização deste trabalho, conseguimos entender melhor como realizar processos em Java, e reforçar a aprendizagem de autómatos, nomeadamente a realização de diagramas de estados, e a maneira de como são implementados.

Com as classes usadas, nomeadamente o `MappedByteBuffer`, e todas as classes referentes à classe `File`, foi possível realizar a comunicação entre os diferentes processos, e conseguir associar esses métodos a realizarem diferentes tarefas, nomeadamente fazer a comunicação entre estes dois processos, e criar um buffer permitindo que diferentes mensagens sejam guardadas em diferentes blocos de memória.

O grupo concretizou a maioria dos objetivos propostos no enunciado, o que facilitou a aprendizagem acerca desta matéria. Inicialmente foi criada uma GUI para o processo em estudo, e de seguida, um diagrama de atividade que conseguisse concretizar o correto funcionamento da aplicação, no final foi tudo testado, e os resultados foram razoáveis ao longo de vários testes realizados.

Resumindo, os objetivos foram alcançados dentro dos nossos conhecimentos, mas ao longo da realização do trabalho, o grupo deparou-se com problemas, e algumas questões durante a realização do trabalho, nomeadamente como poderíamos incrementar o ID de uma mensagem sabendo que estávamos a escrever dentro dum buffer, como saber onde ler ou escrever, e como fazer a comunicação entre os diferentes processos, apesar das dificuldades sentidas, o grupo conseguiu adquirir conhecimentos acerca de toda a nova matéria lecionada, levando mais conhecimentos acerca de processos, que iram a dar jeito nos próximos trabalhos práticos.

## 13 Anexo

Nesta seção encontra-se todo o código realizado acerca do primeiro trabalho prático.

### 13.1 ClasseGui Chat

---

```
public class GuiChat extends JFrame {

    private static final long serialVersionUID = 1L;
    private JTextArea mensagensRecebidasTextArea,
    enviarTextArea,
    fileNameTextArea;
    private JOptionPane messageWarning;
    private JRadioButton
    javaRadioButton,
    todasRadioButton,
    FSORadioButton,
    RobotsRadioButton;
    private final ButtonGroup buttonGroup = new ButtonGroup();
    private JButton abrirBTN,enviarBTN,browseBTN, terminarBTN;
    private CanalComunicacao cc;
    private MaquinaEstados states;
    private int guiID;
    public String FilePath;

    /**
     * Create the application.
     */
    public GuiChat(MaquinaEstados states) {
        this.states = states;
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        guiID+=1;
        getContentPane().setForeground(Color.BLUE);
        setBounds(100, 100, 750, 482);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);
```

```
setTitle("My Chat "+guiID);

try {cc = new CanalComunicacao();} catch (IOException e1)
    {e1.printStackTrace();}

browseBTN = new JButton("Browse");
browseBTN.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        checkFileCC();
    }
});

browseBTN.setFont(new Font("Tahoma", Font.PLAIN, 14));
browseBTN.setBounds(376, 21, 146, 33);
getContentPane().add(browseBTN);

abrirBTN = new JButton("Abrir");
abrirBTN.setEnabled(true);
abrirBTN.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        createCC();
        terminarBTN.setEnabled(true);
    }
});
abrirBTN.setFont(new Font("Tahoma", Font.PLAIN, 14));
abrirBTN.setBounds(551, 21, 146, 33);
getContentPane().add(abrirBTN);

fileNameTextArea = new JTextArea();
fileNameTextArea.setBackground(Color.WHITE);
fileNameTextArea.setForeground(Color.BLACK);
fileNameTextArea.setBounds(189, 27, 172, 27);
fileNameTextArea.setEditable(false);
getContentPane().add(fileNameTextArea);

JLabel caixaCorreioLabel = new JLabel("Caixa de Correio");
caixaCorreioLabel.setFont(new Font("Tahoma", Font.PLAIN, 14));
caixaCorreioLabel.setBounds(10, 24, 121, 27);
getContentPane().add(caixaCorreioLabel);

JLabel mensagemEnviarLabel = new JLabel("Mensagem A Enviar");
mensagemEnviarLabel.setFont(new Font("Tahoma", Font.PLAIN, 14));
mensagemEnviarLabel.setBounds(189, 58, 172, 27);
```

```
getContentPane().add(mensagemEnviarLabel);

enviarBTN = new JButton("Enviar");
enviarBTN.setEnabled(false);
enviarBTN.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        states.numState = states.WRITE_MESSAGE;
    }
});

enviarBTN.setFont(new Font("Tahoma", Font.PLAIN, 14));
enviarBTN.setBounds(551, 92, 146, 33);
getContentPane().add(enviarBTN);

JButton limparBTN = new JButton("Limpar");
limparBTN.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        mensagensRecebidasTextArea.setText("");
    }
});
limparBTN.setFont(new Font("Tahoma", Font.PLAIN, 14));
limparBTN.setBounds(551, 135, 146, 33);
getContentPane().add(limparBTN);

enviarTextArea = new JTextArea();
enviarTextArea.setForeground(Color.BLACK);
enviarTextArea.setBackground(Color.WHITE);
enviarTextArea.setBounds(191, 95, 331, 27);
getContentPane().add(enviarTextArea);

JLabel mensagemRecebidaTextArea = new JLabel("Mensagens Recebidas");
mensagemRecebidaTextArea.setFont(new Font("Tahoma", Font.PLAIN,
    14));
mensagemRecebidaTextArea.setBounds(189, 138, 172, 27);
getContentPane().add(mensagemRecebidaTextArea);

JScrollPane scrollPaneRecebidas = new JScrollPane();
scrollPaneRecebidas.setBounds(189, 178, 508, 169);
getContentPane().add(scrollPaneRecebidas);

mensagensRecebidasTextArea = new JTextArea();
scrollPaneRecebidas.setViewportViewView(mensagensRecebidasTextArea);
mensagensRecebidasTextArea.setForeground(Color.BLACK);
```

```
mensagensRecebidasTextArea.setBackground(Color.WHITE);
mensagensRecebidasTextArea.setEditable(false);

JLabel tipoMensagemLabel = new JLabel("Tipo de Mensagem");
tipoMensagemLabel.setFont(new Font("Tahoma", Font.PLAIN, 14));
tipoMensagemLabel.setBounds(20, 80, 122, 27);
getContentPane().add(tipoMensagemLabel);

todasRadioButton = new JRadioButton("Todas");
todasRadioButton.setSelected(true);
buttonGroup.add(todasRadioButton);
todasRadioButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

    }
});
todasRadioButton.setFont(new Font("Tahoma", Font.PLAIN, 14));
todasRadioButton.setBounds(20, 124, 135, 21);
getContentPane().add(todasRadioButton);

FSOradioButton = new JRadioButton("FSO");
buttonGroup.add(FSOradioButton);
FSOradioButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

    }
});
FSOradioButton.setFont(new Font("Tahoma", Font.PLAIN, 14));
FSOradioButton.setBounds(20, 184, 135, 21);
getContentPane().add(FSOradioButton);

RobotsRadioButton = new JRadioButton("Robots");
buttonGroup.add(RobotsRadioButton);
RobotsRadioButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

    }
});
RobotsRadioButton.setFont(new Font("Tahoma", Font.PLAIN, 14));
RobotsRadioButton.setBounds(20, 254, 135, 21);
getContentPane().add(RobotsRadioButton);

javaRadioButton = new JRadioButton("Java");
```



```
buttonGroup.add(javaRadioButton);
javaRadioButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

    }
});
javaRadioButton.setFont(new Font("Tahoma", Font.PLAIN, 14));
javaRadioButton.setBounds(20, 326, 135, 21);
getContentPane().add(javaRadioButton);

terminarBTN = new JButton("Terminar");
terminarBTN.setEnabled(false);
terminarBTN.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        states.numState = states.TERMINATE;
    }
});
terminarBTN.setFont(new Font("Tahoma", Font.PLAIN, 14));
terminarBTN.setBounds(387, 136, 154, 30);
getContentPane().add(terminarBTN);

JScrollPane pane = new JScrollPane ();
pane.getViewPort ().setView ( mensagemRecebidaTextArea );
pane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

setVisible(true);
}

/**
 * Metodo que verifica o ficheiro do canal de comunicacao
 */
private void checkFileCC() {
    String filename = File.separator+"tmp";
    JFileChooser chooser = new JFileChooser(new File(filename));
    JFrame frame = null;
    int result = chooser.showSaveDialog(frame );
    if(result == JFileChooser.CANCEL_OPTION) {
        messageWarning.showMessageDialog(null, "A opo de cancel foi
            selecionada!", "Warning!", JOptionPane.ERROR_MESSAGE);
    }else {
        File file = chooser.getSelectedFile();
        System.out.println(file);
    }
}
```

```
        FilePath=file.getAbsolutePath();
        fileNameTextArea.append(file.getAbsolutePath());
    }
}

/**
 * Metodo que permite abrir e criar um canal de comunicacao
 */
private void createCC() {
    enviarBTN.setEnabled(true);
    messageWarning.showMessageDialog(null, "Foi criado um canal de
        comunicao!", "Warning!", JOptionPane.ERROR_MESSAGE);
}

public boolean getAbrirButton() {
    return(this.abrirBTN.getModel().isArmed() ? true:false);
}

public boolean isJavaRadio() {
    return javaRadioButton.isSelected();
}

public boolean isTodasRadio() {
    return todasRadioButton.isSelected();
}

public boolean isFSORadio() {
    return FSOradioButton.isSelected();
}

public boolean isRobotsRadio() {
    return RobotsRadioButton.isSelected();
}

public JTextArea getMensagensRecebidasTextArea() {
    return mensagensRecebidasTextArea;
}

/**
 * Mtodo que permite criar uma scrollbar quando sao enviadas muitas
 * mensagens em simultaneo
 * @param mensagensRecebidasTextArea
 */
```

```
public void setMensagensRecebidasTextArea(String
    mensagensRecebidasTextArea) {
    this.mensagensRecebidasTextArea.append(mensagensRecebidasTextArea);
    this.mensagensRecebidasTextArea.setCaretPosition
        (this.mensagensRecebidasTextArea.getDocument().getLength());
}

public JTextArea getEnviarTextArea() {
    return enviarTextArea;
}

public void setEnviarTextArea(JTextArea enviarTextArea) {
    this.enviarTextArea = enviarTextArea;
}
}
```

---

Listing 7: Classe GuiChat

## 13.2 Mensagem

---

```
public class Mensagem {
    private int id, tipo;
    private String texto;
    /**
     * Construtor que necessario para passar o id, o tipo e o texto,
     * quando o
     * outro processo receber a mensagem
     * @param id
     * @param tipo
     * @param texto
     */
    public Mensagem(int id, int tipo, String texto) {
        this.id = id;
        this.tipo = tipo;
        this.texto = texto;
    }

    /**
     * Construtor que inicializa os diferentes atributos
     */
    public Mensagem() {
        id = 0;
        tipo = 0;
        texto = null;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getTipo() {
        return tipo;
    }

    public void setTipo(int tipo) {
        //System.out.println("TOU A METER O TIPO: "+ tipo);
    }
}
```

```
        this.tipo = tipo;
    }

    public String getTexto() {
        return texto;
    }

    public void setTexto(String texto) {
        this.texto = texto;
    }
}
```

---

Listing 8: Classe Mensagem

### 13.3 ICanalComunicaao

---

```
public interface ICanalComunicacao {

    public boolean openChannel(String Filename) throws IOException;

    public void write(Mensagem m) throws IOException;

    public String receberMensagem();

    public void closeChannel();
}
```

---

Listing 9: Interface ICanalComunicaao

## 13.4 CanalComunicacao

```
/**
 * Classe onde realizada a comunicacao entre os processos na mesma mquina
 */
public class CanalComunicacao implements ICanalComunicacao {

    private File ficheiro;
    private MappedByteBuffer buffer;//ira guardar a mensagem
    private RandomAccessFile raf;
    private final int MAX_DIM_BUFFER = 520;// 4+4+256*2
    private int mensagemID, tipo;
    private FileLock fl;
    public FileChannel canal;//permite fazer a ligacao

    public CanalComunicacao() throws IOException {
        canal = null;
        ficheiro = null;
        raf = null;
        buffer = null;
        mensagemID = 0;
        tipo = 0;
    }

    /**
     * Efetua a abertura do canal
     */
    @Override
    public boolean openChannel(String filename) throws IOException{
        ficheiro = new File(filename);
        try {
            this.raf = new RandomAccessFile(this.ficheiro, "rw");
            this.canal = this.raf.getChannel();
            System.out.println("CANAL INICIAL: "+this.canal);
        } catch (FileNotFoundException e) {
            return false;
        }try {
            buffer =
                canal.map(FileChannel.MapMode.READ_WRITE, 0, MAX_DIM_BUFFER);
            System.out.println("BUFFER INICIAL: "+buffer);
        } catch (IOException e) {
            return false;
        }
    }
}
```

```
        return true;
    }

    /**
     * Mtodo que recebe uma mensagem
     * @throws IOException
     */
    @Override
    public String receberMensagem() {
        String msg = new String();
        try {
            fl = canal.lock(0,canal.size(),false);
            char c;

            this.buffer.position(8);
            while ((c=buffer.getChar())!='\0') {
                msg += c;
            }
            fl.release();

        } catch (IOException e) {
            e.printStackTrace();
        }
        return msg;
    }

    /**
     * Mtodo que escreve uma mensagem para dentro do buffer
     * @throws IOException
     */
    @Override
    public void write(Mensagem m) throws IOException {
        System.out.println("CANAL ENVIAR: "+this.canal);
        fl = canal.lock(0,canal.size(),false);
        char c;
        buffer.position(0);
        buffer.putInt(m.getId());

        m.setId(m.getId()+1);
        setMensagemID(m.getId());

        buffer.putInt(m.getTipo());
        for (int i= 0 ; i< m.getTexto().length(); ++i){
```

```
        c= m.getTexto().charAt(i);
        buffer.putChar(c);
    }
    buffer.putChar('\0');

    fl.release();
}

/**
 * Metodo que termina a ligaca do canal de comunicacao
 */
@Override
public void closeChannel() {
    try {
        limpar();
        canal.close();
        raf.close();
    } catch (IOException e) {
        canal = null;
        raf = null;
    }
}

/**
 * Limpa o buffer
 */
public void limpar() {
    buffer.position(0);
    for(int i = 0; i < MAX_DIM_BUFFER;i++) {
        buffer.put(i,(byte) 0);
    }
}

/**
 * Mtodo que permite retornar o id da mensagem dentr do buffer
 * @return o id da mensagem
 */
public int receberID() {
    int id;
    this.buffer.position(0);
    id = buffer.getInt();
    return id;
}
```



```
/**
 * Metodo que retorna o tipo de processo que possvel enviar a mensagem
 * @return o tipo de processo
 */
public int receberTipo() {
    int tipo;
    this.buffer.position(4);
    tipo = buffer.getInt();
    return tipo;
}

public int getMensagemID() {
    return mensagemID;
}

public void setMensagemID(int mensagemID) {
    this.mensagemID = mensagemID;
}

public int getTipo() {
    return tipo;
}

public void setTipo(int tipo) {
    this.tipo = tipo;
}
}
```

---

Listing 10: Classe CanalComunicacao

## 13.5 MaquinaEstados

```
public class MaquinaEstados implements Runnable {
    protected int state,numState;
    private GuiChat gui;
    private CanalComunicacao cc;
    private Mensagem mensagem;
    private int tipoGeral;
    public static final int INITIAL_STATE = 0,
        CREATE_CHANNEL = 1,
        WRITE_MESSAGE = 2,
        READ_MESSAGE = 3,
        TERMINATE_APP = 4;

    public MaquinaEstados() {
        try {
            cc = new CanalComunicacao();
            gui = new GuiChat(this);
            mensagem = new Mensagem();
            numState = INITIAL_STATE;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        while(true) {
            switch(numState) {
                case INITIAL_STATE:
                    System.out.println("ESTOU NO INICIAL");
                    if(gui.getAbrirButton())
                        numState = CREATE_CHANNEL;
                    break;
                case CREATE_CHANNEL:
                    System.out.println("ESTOU NO CREATE");
                    try {
                        if(cc.openChannel(gui.FilePath))
                            numState = READ_MESSAGE;
                    } catch (IOException e2) {
                        e2.printStackTrace();
                    }
            }
        }
    }
}
```

```
        break;
    case WRITE_MESSAGE:
        System.out.println("ENTREI NO WRITE");
        if(gui.isFSORadio()) {
            tipoGeral = 1;
        } else if(gui.isRobotsRadio()) {
            tipoGeral = 2;
        } else if(gui.isJavaRadio()) {
            tipoGeral = 3;
        } else {
            tipoGeral = 0;
        }
        String s;
        s = gui.getEnviarTextArea().getText();
        System.out.println("MENSAGEM ANTERIOR: "+cc.receberID());
        mensagem.setId(cc.receberID()+1);
        mensagem.setTipo(tipoGeral);
        mensagem.setTexto(s);
        try {
            cc.write(mensagem);
        } catch (IOException e) {
            e.printStackTrace();
        }
        numState = READ_MESSAGE;
        break;
    case READ_MESSAGE:
        try {
            Thread.sleep(1000);
            int id = cc.receberID();
            int tipo = cc.receberTipo();
            String texto = cc.receberMensagem();
            if(gui.isTodasRadio()) {
                gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo: "+tipo+" Mensagem: "+texto+ "\n");
            } else if(gui.isFSORadio() && (tipo == 1 || tipo ==0)) {
                gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo: "+tipo+" Mensagem: "+texto+ "\n");
            } else if(gui.isRobotsRadio() && (tipo == 2 || tipo ==0)) {
                gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo: "+tipo+" Mensagem: "+texto+ "\n");
            } else if(gui.isJavaRadio() && (tipo == 3 || tipo ==0)) {
                gui.setMensagensRecebidasTextArea("ID: "+id+" Tipo: "+tipo+" Mensagem: "+texto+ "\n");
            } else {

```

```
        gui.setMensagensRecebidasTextArea("No est a receber  
        nenhuma mensagem \n");  
    }  
    } catch (InterruptedException e1) {e1.printStackTrace();}  
    break;  
case TERMINATE_APP:  
    System.out.println("TERMINATE");  
    cc.closeChannel();  
    break;  
    }  
    }  
}  
public static void main(String[] args) {  
    MaquinaEstados states = new MaquinaEstados();  
    states.run();  
}
```

---

Listing 11: Classe MaquinaEstados

## 14 Bibliografia

Folhas fornecidas pelos engenheiros responsáveis da unidade curricular

Oracle(MappedByteBuffer):

<https://docs.oracle.com/javase/7/docs/api/java/nio/MappedByteBuffer.html>

Oracle(FileChannel):

<https://docs.oracle.com/javase/7/docs/api/java/nio/channels/FileChannel.html>

Processos: [https://pt.wikipedia.org/wiki/Processo\(informática\)](https://pt.wikipedia.org/wiki/Processo(informática))