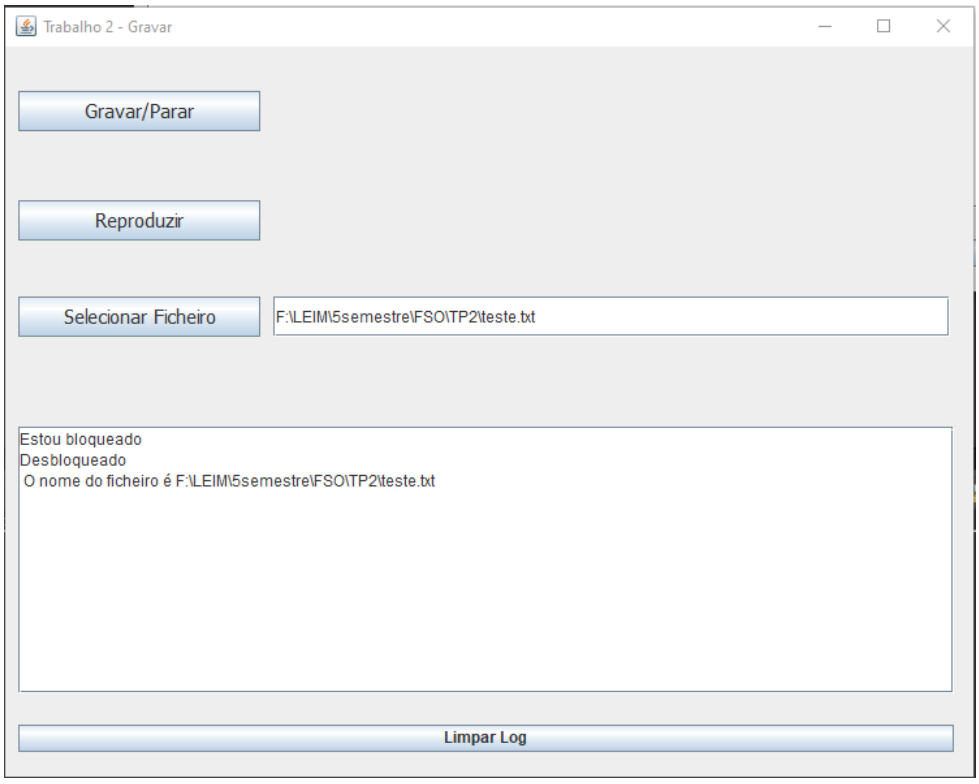
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos - 2324SI

2º Trabalho Prático - Aula prática 4



Docente Carlos Carvalho

Realizado por (Grupo 7):
Diogo Santos 48626
Pedro Silva 48965
João Fonseca 49707

26 de setembro de 2024

Conteúdo

Lista de Figuras

1 Introdução

Esta aula consistiu em desenhar o diagrama de atividades da tarefa GRAVAR. Implementação e teste desta tarefa conjuntamente com a tarefa SUBDITO e com o movimento do robot. A tarefa Gravar tem dois objetivos: gravar as instruções enviadas pelo Rei para o Súbdito num ficheiro de texto e fazer a reprodução desse mesmo ficheiro enviando os comandos para o Súbdito os realizar. A tarefa Gravar está dividida em 3 classes: a BD_Gravar, a GUI_Gravar e a App_Gravar. A classe BD_Gravar vai conter a informação sobre os estados gravar e reproduzir assim como o nome do ficheiro, a GUI_Gravar foi desenvolvida na primeira aula e a App_Gravar vai ter que gerir estas duas funcionalidades da tarefa. Também foram criadas classes auxiliares para ajudar no processo de gravação.

2 Desenvolvimento

2.1 Diagrama de Atividades

Começamos pelo desenvolvimento do diagrama de atividades. Este permite-nos estruturar o resto da aula pois conseguimos estabelecer claramente quais são os objetivos e a forma como opera a tarefa Gravar.

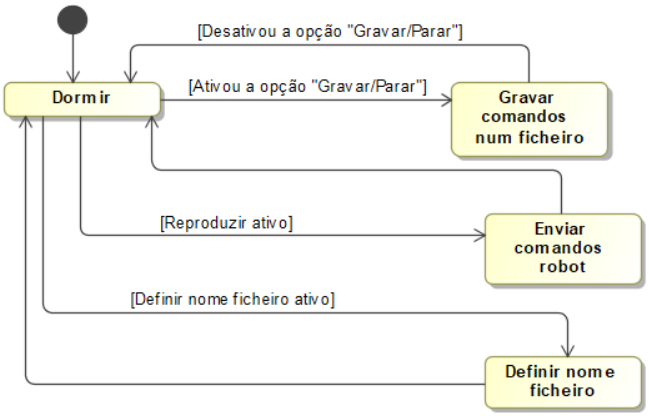


Figura 1: Diagrama de atividades

Como podemos observar na imagem acima, a tarefa Gravar inicia no estado dormir, mantendo-se no mesmo até o utilizador carregar num dos botões. Começando pelo mais importante e pelo qual é obrigatório o utilizador usar ou os outros dois não funcionam, o botão 'Escolher Ficheiro'. Este vai permitir que possamos escolher o local onde vai ser efetuada a gravação e/ou reprodução dos movimentos. Com o ficheiro escolhido podemos premir o botão gravar ou reproduzir não os dois ao mesmo tempo. Se o gravar for escolhido todos os movimentos enviados pelo Rei para o Súbdito ficarão guardados no tal ficheiro. Se o reproduzir for escolhido, todas as ordens contidas no ficheiro irão ser enviadas pela tarefa Gravar para a tarefa Súbdito.

2.2 Tarefa Gravar

2.2.1 Gravar

Vamos começar com a gravação dos comandos e dos ficheiros. Para facilitar esse processo vamos criar um "robot espião"em que o objetivo é, cada vez que o robot for chamado para realizar um comando vai existir uma verificação para saber se é suposto guardar essa ordem no ficheiro ou não. Se for suposto gravar a classe "Gravador"trata do resto, se não apenas é efetuado o comando. Fica assim todo o processo:

```
synchronized void reta(Mensagem msg)
{
    super.reta(msg);
    robot.Reta(msg.getArg1());
}
```

Figura 2: Método 'reta' - Classe myRobotLegoEV3

Este método é chamado quando o súbdito quer enviar uma ordem para o robot realizar. Depois de entrarmos vamos logo para o método com o mesmo nome que este mas da classe "pai", esta classe estende RobotEV3, quando voltar enviamos a ordem para o robot normalmente. O método "reta"da classe estendida fica assim:

```
synchronized void reta(Mensagem msg) {
    System.out.println(gravador.getGravar());
    if (gravador!=null && gravador.getGravar())
        gravador.reta(msg);
    //System.currentTimeMillis();
}
```

Figura 3: Método 'reta' - Classe RobotEV3

Esta classe vai então receber uma instância da classe "Gravador", classe que trata de guardar a informação, no método vamos primeiro verificar se existe mesmo uma instância da classe mencionada e se o botão gravar foi premido, se estes forem os casos vamos então para o método com o mesmo nome deste mas da classe de gravação.

```
void reta(Mensagem msg)
{
    // Try-with-resources to automatically close resources (like FileWriter)
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(bdGravar.getNome(), true))) {
        // Write data to the file
        writer.write(
                ""+ msg.tipo + "," +
                "" + msg.arg1 + "," +
                "" + System.currentTimeMillis() + "" + EOL);

        System.out.println("Data has been successfully saved to the file.");

    } catch (IOException e) {
        // Handle exceptions
        System.out.println("No file.");
        e.printStackTrace();
    }
}
```

Figura 4: Método 'reta' - Classe Gravador

Neste método, recebendo a mensagem, vamos gravar a mensagem no ficheiro, guardado na base de dados. As mensagens no ficheiro vão ter o seguinte formato: 'tipo,argumento1,argumento2,currentTimeMillis()'. O tipo permite-nos identificar qual o movimento que o robot vai realizar, os argumentos vão ser os dados do movimento e o currentTimeMillis vai nos dar o tempo de quando foi realizada a instrução. O tempo vai ser necessário para simularmos corretamente o conjunto de instruções dadas pelo Rei ao Súbdito, de tal forma a que se houver uma pausa de 5 segundos entre comandos isto irá refletir-se na reprodução do ficheiro.

2.2.2 Reproduzir

O botão reproduzir tem como reproduzir as instruções guardadas no ficheiro de texto escolhido. O envio dessas instruções da tarefa Gravar para a tarefa Súbdito vai funcionar com o mesmo mecanismo que a comunicação entre o Rei e o Súbdito, ou seja, através dum buffer circular. O estado reproduzir da tarefa fica assim:

```
case reproduzir:
    String filePath = bd.getNome();
    long currentTime = 0, lastTime = 0;
    // Try-with-resources to automatically close resources (like BufferedReader)
    try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
        String line;
        int numLine = 0;  // Initialize the line number
        while ((line = reader.readLine()) != null) {
              // Increment the line number for each line read

            // Split the line into parts using a delimiter (assuming it's a CSV format)
            String[] parts = line.split(",");

            // Assuming the first part is an integer
            int tipo = Integer.parseInt(parts[0]);

            switch (tipo) {
                case 1:
                    // Assuming numLine is declared and initialized somewhere in your code
                    msg = new MensagemReta(numLine, tipo, Integer.parseInt(parts[1]));
                    currentTime = Long.parseLong(parts[2]);
                    bd.addMensagem(msg);
                    break;
```

Figura 5: Estado 'reproduzir' 1/2 - Classe App_Gravar

Vamos começar por inicializar as nossas variáveis como o nome do ficheiro, duas variáveis de tempo para fazermos o controlo do tempo entre instruções e o número da linha, que vai corresponder ao id. De seguida vamos lendo cada linha do ficheiro e guardar cada parte, divididas por ',', num array. Descobrimos qual o tipo da mensagem, transformamos numa mensagem correspondente e guardamos a numa lista de mensagens para enviar.

```
try {

    ocupadaMyMensagem.acquire();
    acessoMyMensagem.acquire();

    long sleepTime = currentTime - lastTime;
    //System.out.println(sleepTime);
    // Check if the sleep time is non-negative
    if (lastTime!= 0) {
        // Introduce a delay based on the time diffe
        System.out.println("sleep");
        Thread.sleep(sleepTime);
        System.out.println("wake");
        if (!bd.getReproduzir())
            break;
    }

} catch (InterruptedException e) {}
    bufferCircular.inserirElemento(myMensagem);
    acessoMyMensagem.release();
    livreMyMensagem.release();
    haTrabalho.release();
    if(tipo != 4)
        gui.txtLog.append("Enviei = " + msg + "\n");

    bd.removeMensagem();
    lastTime = currentTime;
    numLine++;
```

Figura 6: Estado 'reproduzir' 2/2 - Classe App_Gravar

Depois antes de prosseguirmos com o envio da mensagem para o buffer circular vamos verificar qual foi o tempo entre os comandos. Para isso pegamos nos currentTimeMillis da linha atual e subtraímos com a anterior, se for a primeira isto não acontece, aplicando um Thread.sleep() com o tempo correspondente. Estas variáveis são guardadas em formato "long"devido ao seu tamanho. Continuamos da mesma forma que a tarefa Rei e no fim passamos o currentTime para lastTime de forma a dar continuidade ao ciclo.

3 Conclusões

Depois de algumas dicas do docente em como proceder no desenvolvimento da tarefa, principalmente o gravar, o processo tornou-se mais fácil do que aparentava. Conseguimos colocar todas as funcionalidades a trabalhar incluindo semáforos nos locais em que estavam a gastar processador sem qualquer necessidade, como por exemplo: na App_Gravar introduzimos um semáforo no estado gravar pois não é aqui que se faz a gravação dos ficheiros logo não existe necessidade desta tarefa estar a gastar recursos. Com isto ficamos com todas as partes da aplicação a funcionar como pretendido.

4 Bibliografia

1. Folhas de Computação Física - Jorge Pais, 2023/2024

5 Código Java GUI_Subdito, BD_Subdito, App_Subdito

```
1
2  Classe GUI_Gravar
3
4  package ptrabalho;
5
6  import java.awt.EventQueue;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import java.io.BufferedWriter;
10 import java.io.FileWriter;
11 import java.io.IOException;
12 import java.nio.file.Files;
13 import java.nio.file.Path;
14 import java.nio.file.Paths;
15 import java.nio.file.StandardOpenOption;
16
17 import javax.swing.JFileChooser;
18 import javax.swing.JFrame;
19 import javax.swing.JPanel;
20 import javax.swing.border.EmptyBorder;
21 import javax.swing.JToggleButton;
22 import java.awt.Font;
23 import javax.swing.JTextField;
24 import javax.swing.JButton;
25
26 public class GUI_Gravar extends GUI_Base {
27
28     private JPanel contentPane;
29     private JTextField txtFile;
30     protected JToggleButton tglGravar;
31     protected JToggleButton tglReproduzir;
32     protected JButton btnFile;
33
34
35     public GUI_Gravar(BD_Gravar bd)
36   {
37     super(bd);
38     EventQueue.invokeLater(new Runnable()
39     {
40       public void run()
41       {
42         try
43         {
44           init_Gravar(bd);
45
46         } catch (Exception e)
47         {
48           e.printStackTrace();
49         }
50       }
51     });
52   }
53
54   public void init_Gravar(BD_Gravar bd) {
55
56     setTitle("Trabalho 2 - Gravar");
57       //setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
58       setBounds(1000, 400, 760, 600);
59       txtFile = new JTextField();
```

```
60        txtFile.setBounds(206, 192, 520, 31);
61        getContentPane().add(txtFile);
62        txtFile.setColumns(10);
63
64        tglGravar = new JToggleButton("Gravar/Parar");
65        tglGravar.setEnabled(false);
66        tglGravar.setFont(new Font("Tahoma", Font.PLAIN, 15));
67        tglGravar.setBounds(10, 34, 186, 31);
68        getContentPane().add(tglGravar);
69
70        tglReproduzir = new JToggleButton("Reproduzir");
71        tglReproduzir.setEnabled(false);
72        tglReproduzir.setFont(new Font("Tahoma", Font.PLAIN, 15));
73        tglReproduzir.setBounds(10, 118, 186, 31);
74        getContentPane().add(tglReproduzir);
75
76        btnFile = new JButton("Selecionar Ficheiro");
77        btnFile.setEnabled(false);
78        btnFile.setFont(new Font("Tahoma", Font.PLAIN, 15));
79        btnFile.setBounds(10, 192, 186, 31);
80        getContentPane().add(btnFile);
81        btnFile.addActionListener(new ActionListener()
82        {
83          public void actionPerformed(ActionEvent e)
84          {
85            JFileChooser fileChooser = new JFileChooser(System.
    getProperty("user.dir"));
86                    if (fileChooser.showSaveDialog(null) ==
    JFileChooser.APPROVE_OPTION)
87                    {
88                      String file = fileChooser.getSelectedFile().
    getAbsolutePath();
89                      bd.setNome(file);
90                      write(" O nome do ficheiro é " + file + "\n");
91                      txtFile.setText(file);
92                    }
93             }
94         });
95
96        txtFile.addActionListener(new ActionListener()
97     {
98        public void actionPerformed(ActionEvent e)
99        {
100         bd.setNome(txtFile.getText());
101         write(" O nome do ficheiro é " + bd.getNome() + "\n");
102       }
103    });
104
105       tglGravar.addActionListener(new ActionListener() {
106           public void actionPerformed(ActionEvent e) {
107               // Check if the toggle button is selected (on)
108               boolean isOn = tglGravar.isSelected();
109
110               // Set the 'gravar' boolean based on the toggle
    button state
111               try {
112           bd.setGravar(isOn);
113         } catch (InterruptedException e1) {
114           // TODO Auto-generated catch block
115           e1.printStackTrace();
```

```
116         }
117               if(isOn) {
118                 // Create an empty byte array
119                   byte[] emptyBytes = new byte[0];
120 
121                   // Open the file in write mode and overwrite it
    with an empty byte array
122 
123                   Path path = Paths.get(bd.getNome());
124                   try {
125             Files.write(path, emptyBytes, StandardOpenOption.WRITE,
     StandardOpenOption.TRUNCATE_EXISTING);
126           } catch (IOException e1) {
127             // TODO Auto-generated catch block
128             e1.printStackTrace();
129           }
130               }
131               else
132               {
133                 // Try-with-resources to automatically close
    resources (like FileWriter)
134                   try (BufferedWriter writer = new BufferedWriter(
    new FileWriter(bd.getNome(), true))) {
135                       // Write data to the file
136                       writer.write("5" + System.lineSeparator());
137                       System.out.println("Data has been
    successfully saved to the file.");
138                   } catch (IOException e1) {
139                       // Handle exceptions
140                       e1.printStackTrace();
141                   }
142               }
143           }
144       });
145 
146       tglReproduzir.addActionListener(new ActionListener() {
147         public void actionPerformed(ActionEvent e) {
148             // Check if the toggle button is selected (on)
149             boolean isOn = tglReproduzir.isSelected();
150 
151             // Set the 'gravar' boolean based on the toggle button
    state
152             bd.setReproduzir(isOn);
153 
154             // Append a message to the 'txtLog' text component
155             if(isOn)
156               write(" Começou a reproduzir \n");
157             else
158               write(" Parou de reproduzir \n");
159         }
160     });
161 
162 
163       setVisible(true);
164     }
165 
166   protected void start()
167   {
168     btnFile.setEnabled(true);
169     tglReproduzir.setEnabled(true);
```

```
170     tglGravar.setEnabled(true);
171   }
172
173   protected void off()
174   {
175     btnFile.setEnabled(false);
176     tglReproduzir.setEnabled(false);
177     tglGravar.setEnabled(false);
178   }
179 }
180
181
182 Classe BD_Gravar
183
184 package ptrabalho;
185
186 import java.util.concurrent.Semaphore;
187
188 public class BD_Gravar extends BD_Base {
189
190
191   private String nome;
192   private boolean gravar, reproduzir;
193   private Semaphore gravarS = new Semaphore(0);
194
195   public BD_Gravar()
196     {
197       super();
198
199     }
200
201   public void setNome(String n)
202     {
203         nome = n;
204     }
205
206     public String getNome()
207     {
208         return nome;
209     }
210
211     public void setGravar(boolean g) throws InterruptedException
212     {
213         gravar = g;
214         if (!g)
215           gravarS.release();
216
217     }
218
219     public boolean getGravar()
220     {
221         return gravar;
222     }
223
224     public Semaphore getGravarS()
225     {
226       return gravarS;
227     }
228
229     public void setReproduzir(boolean r)
```

```
230     {
231         reproduzir = r;
232     }
233 
234     public boolean getReproduzir()
235     {
236         return reproduzir;
237     }
238 }
239 
240 
241 Classe App_Gravar
242 
243 package ptrabalho;
244 
245 import java.io.BufferedReader;
246 import java.io.FileReader;
247 import java.io.IOException;
248 import java.util.concurrent.Semaphore;
249 
250 public class App_Gravar extends Thread {
251 
252   private Gravador gravador;
253   protected GUI_Gravar gui;
254     private BD_Gravar bd;
255     Mensagem msg;
256     Mensagem myMensagem = null;
257     private int state = 2;
258     private int counter = 0;
259     private final int reproduzir = 1;
260     private final int dormir = 2;
261     private final int gravar = 3;
262     private final int bloqueado = 4;
263     private final int ler = 5;
264 
265     BufferCircular bufferCircular;
266     private Semaphore haTrabalho, livreMyMensagem,
    ocupadaMyMensagem, acessoMyMensagem, gravarAvailable;
267 
268   public App_Gravar(BD_Gravar bg, Gravador g, Semaphore ga,
     BufferCircular bc, Semaphore ht)
269     {
270       gravador = g;
271         gui = new GUI_Gravar(bg);
272         bd = bg;
273         bufferCircular= bc;
274         haTrabalho = ht;
275         gravarAvailable = ga;
276         livreMyMensagem= new Semaphore(1);
277       ocupadaMyMensagem= new Semaphore(0);
278       acessoMyMensagem= new Semaphore(1);
279 
280     }
281 
282   public void setMensagem(Mensagem m)
283     {
284       try {
285       livreMyMensagem.acquire();
286       acessoMyMensagem.acquire();
287       } catch (InterruptedException e) {}
```

```
288       myMensagem= m;
289       acessoMyMensagem.release();
290       ocupadaMyMensagem.release();
291     }
292 
293   public void run()
294     {
295         while(true) {
296 
297           switch (state) {
298 
299             case dormir:
300               //System.out.println("sleep");
301               try {
302                 Thread.sleep(100);
303                 //System.out.println("permits" + reiAvailable.
    availablePermits());
304               if(gravarAvailable.availablePermits() == 0) {
305                 state = bloqueado;
306                 break;
307               }
308             } catch (InterruptedException e) {
309               // TODO Auto-generated catch block
310               e.printStackTrace();
311             }
312             if (bd.getGravar())
313               state = gravar;
314             if(bd.getReproduzir())
315               state = reproduzir;
316             break;
317 
318 
319             case gravar:
320               System.out.println(bd.getGravarS().availablePermits()
    );
321               gui.write("Estou a Gravar \n");
322               try {
323             bd.getGravarS().acquire();
324           } catch (InterruptedException e) {
325             // TODO Auto-generated catch block
326             e.printStackTrace();
327           }
328             gui.write("Parei de gravar \n");
329             state = dormir;
330             break;
331 
332             case reproduzir:
333                 String filePath = bd.getNome();
334                 long currentTime = 0, lastTime = 0;
335                 // Try-with-resources to automatically close
    resources (like BufferedReader)
336                 try (BufferedReader reader = new BufferedReader(new
     FileReader(filePath))) {
337                     String line;
338                     int numLine = 0;  // Initialize the line number
339                     while ((line = reader.readLine()) != null) {
340                         // Increment the line number for each
    line read
341 
```

```
342                         // Split the line into parts using a
    delimiter (assuming it's a CSV format)
343                         String[] parts = line.split(",");
344 
345                         // Assuming the first part is an integer
346                         int tipo = Integer.parseInt(parts[0]);
347 
348                         switch (tipo) {
349                             case 1:
350                                 // Assuming numLine is declared and
     initialized somewhere in your code
351                                 msg = new MensagemReta(numLine,
    tipo, Integer.parseInt(parts[1]));
352                                 currentTime = Long.parseLong(parts
    [2]);
353                                 bd.addMensagem(msg);
354                                 break;
355 
356                             case 2:
357                                 // Assuming numLine is declared and
     initialized somewhere in your code
358                                 msg = new MensagemCurvar(numLine,
    tipo, Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
359                                 currentTime = Long.parseLong(parts
    [3]);
360                                 bd.addMensagem(msg);
361                                 break;
362 
363                             case 3:
364                                 // Assuming numLine is declared and
     initialized somewhere in your code
365                                  msg = new MensagemCurvar(numLine,
    tipo, Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
366                                 currentTime = Long.parseLong(parts
    [3]);
367                                 bd.addMensagem(msg);
368                                 break;
369 
370                             case 4:
371                               msg = new MensagemParar(numLine, tipo
    , false);
372                                 //currentTime = Long.parseLong(
    parts[2]);
373                                 bd.addMensagem(msg);
374                                 break;
375 
376                             case 5:
377                               msg = new MensagemVazia(numLine, 5);
378                               bd.addMensagem(msg);
379                                 break;
380                             // Handle other cases if needed
381                         }
382 
383 
384                         if(tipo != 4)
385                           gui.write("Li: " + msg + "\n");
386                         System.out.println("escreve");
387                       System.out.println("Mensagens à espera: " +
    bd.getMensagens().size());
388                       msg = bd.getMensagens().get(0);
```

```
389                         setMensagem(msg);
390                         try {
391
392                           ocupadaMyMensagem.acquire();
393                           acessoMyMensagem.acquire();
394
395                           long sleepTime = currentTime - lastTime;
396                           //System.out.println(sleepTime);
397                             // Check if the sleep time is non-
    negative
398                             if (lastTime!= 0) {
399                                 // Introduce a delay based on the
    time difference
400                               System.out.println("sleep");
401                                 Thread.sleep(sleepTime);
402                                 System.out.println("wake");
403                                 if (!bd.getReproduzir())
404                                   break;
405                             }
406
407                         } catch (InterruptedException e) {}
408                           bufferCircular.inserirElemento(myMensagem);
409                           acessoMyMensagem.release();
410                           livreMyMensagem.release();
411                           haTrabalho.release();
412                           if(tipo != 4)
413                             gui.txtLog.append("Enviei = " + msg + "\n
    ");
414
415                           bd.removeMensagem();
416                           lastTime = currentTime;
417                           numLine++;
418
419                       }
420                   } catch (IOException | NumberFormatException e) {
421                       // Handle exceptions
422                       System.out.println("Error reading or parsing
    the file: " + e.getMessage());
423                       e.printStackTrace();
424                   }
425                   state = dormir;
426                   bd.setReproduzir(false);
427                   gui.tglReproduzir.setSelected(false);
428                   break;
429
430
431               case bloqueado:
432                 try {
433                   gui.write("Estou bloqueado \n");
434                   gravarAvailable.acquire();
435                   gui.write("Desbloqueado \n");
436                   state = dormir;
437                   break;
438                 } catch (InterruptedException e) {
439                   // TODO Auto-generated catch block
440                   e.printStackTrace();
441                 }
442
443       }
444     }
```

```
445
446     }
447 }
448
449
450
451 Classe Gravador
452
453 package ptrabalho;
454
455 import java.io.BufferedWriter;
456 import java.io.FileWriter;
457 import java.io.IOException;
458
459 public class Gravador {
460
461
462   private boolean gravar;
463   private BD_Gravar bdGravar;
464   private final String EOL = System.lineSeparator();
465
466   public Gravador(BD_Gravar bd)
467   {
468     gravar = false;
469     bdGravar = bd;
470   }
471
472   public void setGravar(boolean g)
473   {
474     gravar = g;
475   }
476
477   public boolean getGravar()
478   {
479     return bdGravar.getGravar();
480   }
481
482   void reta(Mensagem msg)
483   {
484     // Try-with-resources to automatically close resources (like
    FileWriter)
485         try (BufferedWriter writer = new BufferedWriter(new
    FileWriter(bdGravar.getNome(), true))) {
486             // Write data to the file
487             writer.write(
488                 ""+ msg.tipo + "," +
489                 "" + msg.arg1 + "," +
490                 "" + System.currentTimeMillis() + "" + EOL);
491
492             System.out.println("Data has been successfully saved to
    the file.");
493
494         } catch (IOException e) {
495             // Handle exceptions
496           System.out.println("No file.");
497             e.printStackTrace();
498         }
499   }
500
501   void curvarDireita(Mensagem msg)
```

```
  {
    // Try-with-resources to automatically close resources (like FileWriter)
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(bdGravar.getNome(), true))) {
            // Write data to the file
            writer.write(
                "" + msg.tipo + "," +
                "" + msg.arg1 + "," +
                "" + msg.arg2 + "," +
                "" + System.currentTimeMillis() + "" + EOL);

            System.out.println("Data has been successfully saved to the file.");
        } catch (IOException e) {
            // Handle exceptions
            e.printStackTrace();
        }
  }

  void curvarEsquerda(Mensagem msg)
  {
    // Try-with-resources to automatically close resources (like FileWriter)
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(bdGravar.getNome(), true))) {
            // Write data to the file
            writer.write(
                "" + msg.tipo + "," +
                "" + msg.arg1 + "," +
                "" + msg.arg2 + "," +
                "" + System.currentTimeMillis() + "" + EOL);

            System.out.println("Data has been successfully saved to the file.");
        } catch (IOException e) {
            // Handle exceptions
            e.printStackTrace();
        }
  }

  void parar(Mensagem msg)
  {
    // Try-with-resources to automatically close resources (like FileWriter)
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(bdGravar.getNome(), true))) {
            // Write data to the file
            writer.write("4" + EOL);
            System.out.println("Data has been successfully saved to the file.");
        } catch (IOException e) {
            // Handle exceptions
            e.printStackTrace();
        }
  }
}


Classe RobotEV3
```

```
553
554 package ptrabalho;
555
556 public class RobotEV3 {
557
558   protected Gravador gravador;
559
560   public RobotEV3(Gravador g) {
561     gravador = g;
562   }
563
564
565   synchronized void reta(Mensagem msg) {
566     //System.out.println(gravador.getGravar());
567     if (gravador!=null && gravador.getGravar())
568       gravador.reta(msg);
569     //System.currentTimeMillis();
570   }
571
572   synchronized void curvarEsquerda(Mensagem msg) {
573     //System.out.println(gravador.getGravar());
574     if (gravador!=null && gravador.getGravar())
575       gravador.curvarEsquerda(msg);
576     //System.currentTimeMillis();
577   }
578
579   synchronized void curvarDireita(Mensagem msg) {
580     //System.out.println(gravador.getGravar());
581     if (gravador!=null && gravador.getGravar())
582       gravador.curvarDireita(msg);
583     //System.currentTimeMillis();
584   }
585
586   synchronized void parar(Mensagem msg) {
587     //System.out.println(gravador.getGravar());
588     if (gravador!=null && gravador.getGravar())
589       gravador.parar(msg);
590     //System.currentTimeMillis();
591   }
592
593 }
594
595
596 Classe myRobotLegoEV3
597
598 package ptrabalho;
599
600 import robot.RobotLegoEV3;
601
602 public class myRobotLegoEV3 extends RobotEV3{
603
604   private RobotLegoEV3 robot;
605
606   public myRobotLegoEV3(Gravador g) {
607     super(g);
608     robot = new RobotLegoEV3();
609   }
610
611
612   public RobotLegoEV3 getRobot() {
```

```
613       return robot;
614     }
615
616
617     synchronized void reta(Mensagem msg)
618     {
619       super.reta(msg);
620       robot.Reta(msg.getArg1());
621     }
622
623     synchronized void curvarDireita(Mensagem msg)
624     {
625       super.curvarDireita(msg);
626       robot.CurvarDireita(msg.getArg1(),msg.getArg2());
627     }
628
629     synchronized void curvarEsquerda(Mensagem msg)
630     {
631       super.curvarEsquerda(msg);
632       robot.CurvarEsquerda(msg.getArg1(),msg.getArg2());
633
634     }
635
636     synchronized void parar(Mensagem msg)
637     {
638       super.parar(msg);
639       robot.Parar(false);
640     }
641 }
```