Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos - 2324SI

1º Trabalho Prático - Aula prática 5

Docente Carlos Carvalho

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

26 de setembro de 2024

Conteúdo

1 Introdução

Esta aula consistiu em desenhar o diagrama de atividades, implementar e testar o processo Rei. O Rei, como no jogo "O Rei Manda"vai enviar instruções, através do canal de comunicação previamente desenvolvido, para o Súbdito em que este vai ter que as realizar. O processo Rei está dividido em 3 classes: a BD_Rei, a GUI_Rei e a App_Rei. As bases destas 3 classes já tinham sido criadas mas nesta aula vamos concretizá-las. A BD_Rei vai extender da BD_Base, ou seja, permite-nos ter uma base de dados apenas para o processo Rei. O mesmo se passa para a GUI_Rei que vai extender da GUI_Base. A App_Rei vai conter uma máquina de estados que vai controlar o envio de mensagens para o canal de comunicação.

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 o processo Rei .

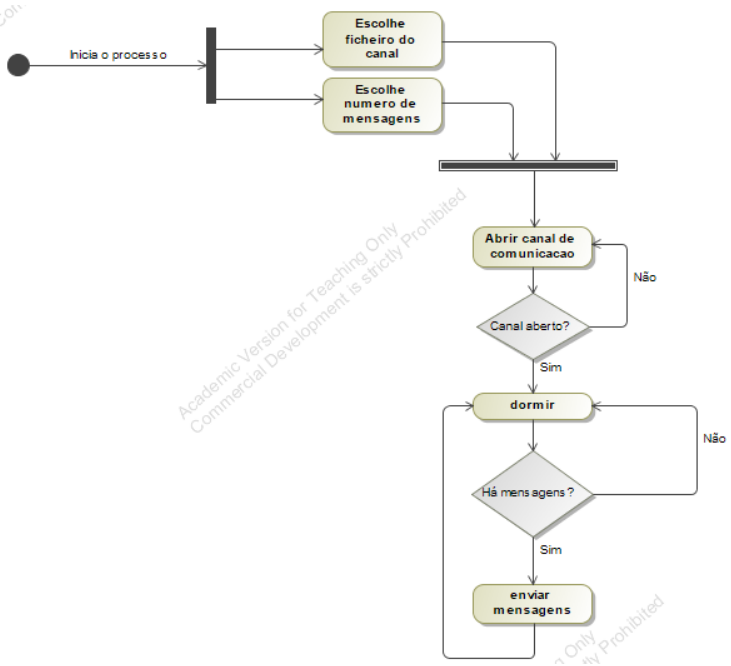

Figura 1: Diagrama de atividades

Como podemos observar na imagem acima, o processo Rei tem como base uma máquina de estados que inicia quando o canal de comunicação estiver aberto. Esta entra num estado de dormir até que existam mensagens que o Rei queira enviar, passando assim para o estado de envio. Neste estado o Rei tenta enviar uma mensagem sendo este envio apenas possível se o buffer não estiver cheio. Se este for o caso a mensagem é enviada e retirada da lista de mensagens e voltamos ao estado dormir. Se não for o caso voltamos ao dormir e repetimos a verificação.

2.2 Processo Rei

2.2.1 BD_Rei

Depois de termos criado o diagrama de atividades vamos atualizar as 3 classes do processo Rei. Na classe BD_Rei criámos uma lista de mensagens assim como um método para adicionar uma mensagem e um método para devolver todas as mensagens. Esta lista assim como os seus métodos vão permitir ao Rei armazenar as instruções que vão ser posteriormente enviadas ao Súbdito.

2.2.2 GUI_Rei

Na classe GUI_Rei vamos alterar os botões para que estes possam criar mensagens e armazená-las na lista mencionada previamente. Também vamos implementar os botões de 8 e 16 mensagens aleatórias. Estes botões vão criar mensagens aleatórias uma a uma usando o método gerarRandomMensagem() que por si usa o método gerarVariáveis().

```
private Mensagem gerarRandomMensagem() {
    Mensagem m = new Mensagem();
    Random rn = new Random();
    int[] variaveis = new int[4]; // array de 4 variaveis
    int tipoMensagem = rn.nextInt(3); // random between 0-2
    if(id==8)
        id=0;
    variaveis = gerarVariaveis(tipoMensagem);
    m.setId(variaveis[0]);
    m.setTipo(variaveis[1]);
    m.setArg1(variaveis[2]);
    m.setArg2(variaveis[3]);
    return m;
}
```

Figura 2: Método gerarRandomMensagem()

```
/*
 * Metodo auxiliar ao gerarRandomMensagem()
 *
 * @param tMsg - tipo de mensagem
 */
private int[] gerarVariaveis(int tMsg) {
    Random rn = new Random();
    int[] variaveis = new int[4];
    int variavel;
    if (tMsg == 0) { // para tMsg 0 faz reta
        variaveis[0] = id;
        variaveis[1] = 1; // 1 equivale a reta na minha mensagem
        variavel = rn.nextInt(45) + 5; // random between 5-50 cm reta
        variaveis[2] = variavel;
        variaveis[3] = 0;
    } else if (tMsg == 1) { // para tMsg 1 faz curva direita
        variaveis[0] = id;
        variaveis[1] = 2; // 2 equivale a reta na minha mensagem
        variavel = rn.nextInt(30); // random between 0-30 raio
        variaveis[2] = variavel;
        variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
        variaveis[3] = variavel;
    } else { // para tMsg 0 faz curva esquerda
        variaveis[0] = id;
        variaveis[1] = 3; // 3 equivale a reta na minha mensagem
        variavel = rn.nextInt(30); // random between 0-30 raio
        variaveis[2] = variavel;
        variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
        variaveis[3] = variavel;
    }
    return variaveis;
}
```

Figura 3: Método gerarVariáveis()

O método gerarRandomMensagem() começa por gerar um número inteiro entre 0 e 2 que vai decidir qual o tipo de mensagem a ser criada. Com este número chamamos o método seguinte, gerarVariáveis(), que nos vai dar valores aleatórios de acordo com o tipo de mensagem escolhida. Se a mensagem escolhida for uma reta a distância percorrida vai ser entre 5 e 50 cm, se for uma das curvas o raio vai ser um valor entre 0 e 30 cm enquanto que o ângulo vai ser entre 20^{o} e 90^{o}. Por fim estes valores são guardados na mensagem e esta na lista de mensagens a enviar.

2.2.3 App_Rei

Finalmente vamos desenvolver a máquina de estados concebida no diagrama de atividades. Como descrito anteriormente esta vai nos permitir fazer o controlo e envio de mensagens no canal de comunicação.

```
public void run() throws InterruptedException
{
    while(!bd.getTerminar()) {

        switch (state) {

            case escreverMensagem:
                System.out.println("escreve");
                System.out.println("Mensagens à espera: " + bd.getMensagens().size());
                msg = bd.getMensagens().get(0);
                msg.setId(counter);
                if (bd.getCanal().GetandSetWrite(msg)) {
                    gui.txtLog.append(" Enviei = " + msg + "\n");
                    counter =++ counter % bd.getNMensagens();
                    bd.removeMensagem();
                    state = esperarTempoExecucao;
                    break;
                }
                else {
                    state = dormir;
                    break;
                    }

            case dormir:
                System.out.println("sleep");
                Thread.sleep(1000);
                if(bd.getMensagens().size()!=0) {
                    state = escreverMensagem;
                    break;
                }
                else {break;}
```

Figura 4: Máquina de Estados

A máquina começa no estado dormir onde após uma espera de 1 segundo verifica se existem mensagens na lista de espera, se este for o caso passamos para o estado escreverMensagem. Neste estado guardamos a primeira mensagem da lista e mudamos o seu id para o correspondente de acordo com a sua posição no buffer. Para controlarmos este id temos um contador que incrementa 1 a cada mensagem enviada a não ser que este valor corresponda ao número de máximo de mensagens no buffer, neste caso o contador volta para 0. De seguida, se o buffer não estiver cheio escrevemos na consola da GUI a mensagem, incrementamos o contador, removemos a mensagem da lista das por enviar e mudamos de estado de volta para dormir.

3 Conclusões

Depois de termos passado a semana inteira a afinar o canal de comunicação e a preparar o processo do Rei conseguimos com que o processo funcionasse sem qualquer problema. O único percalço foi o da GUI travar cada vez que o Rei cria enviar mensagens mas o buffer estava cheio. Isto deve-se ao facto de a máquina de estados ter estado ligada ao pressionar dos botões, o que, em retrospetiva, não foi uma boa ideia pois fazia com que a interface estivesse dependente do envio das mensagens o que não era pretendido.

4 Bibliografia

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

5 Código Java

```
Classe App_Subdito

package ptrabalho;

public class App_Subdito
{
    @SuppressWarnings("unused")
    private GUI_Subdito gui;
    private BD_Subdito bd;
    Mensagem msg = null;
    private int state = 2;
    private int counter = 0;
    private final int receberMensagem = 1;
    private final int dormir = 2;
    private final int esperarTempoExecucao = 3;

    public App_Subdito()
    {
        bd = new BD_Subdito();
        gui = new GUI_Subdito(bd);
    }

    public BD_Subdito getBD()
    {
        return bd;
    }

    public void run() throws InterruptedException
    {
        while(!bd.getTerminar())
          while(!bd.getTerminar()) {

              switch (state) {

              case receberMensagem:
                System.out.println("recebe");
                gui.txtLog.append(" Recebi = " + msg + "\n");
                //counter =++ counter % 8;
                bd.addMensagem(msg);
                state = dormir;
                break;


          case dormir:
            System.out.println("busca");

            if(bd.getCanal() != null){
              msg = bd.getCanal().GetandSetReadLeitor();
              if(msg.getTipo() != 4) {
                System.out.println("existe msg");

                state = receberMensagem;
                break;
              }
              else {
                System.out.println("sleep");
                Thread.sleep(1000);
                break;}
              }else {
```

```
                Thread.sleep(1000);
                System.out.println("sleep");
                break;}

          case esperarTempoExecucao:
            //System.out.println("esperaExec" + id);
            if (msg.getTipo() == 1) {
              //Thread.sleep((long) ((msg.getArg1()) / 0.03));
            } else if (msg.getTipo() == 2 || msg.getTipo() == 3) {
              //Thread.sleep((long) ((msg.getArg1() * (msg.getArg2
    () * 0.017)) / 0.03));
            } else {
              //Thread.sleep(1000);
            }
            state = dormir;
            break;
          }
        }
        System.out.println("sai");
            Thread.sleep(100);
    }

    public static void main(String[] args) throws
    InterruptedException
    {
        App_Subdito app = new App_Subdito();
        System.out.println("A aplicação começou.");
        app.run();
        System.out.println("A aplicação terminou.");
    }

}

Classe BD_Subdito

package ptrabalho;
import robot.RobotLegoEV3;

package ptrabalho;
//import robot.RobotLegoEV3;

public class BD_Subdito extends BD_Base
{
    //private RobotLegoEV3 robot;
    private boolean terminar;
    private boolean ligado;

    private String nome;

    public BD_Subdito()
    {
      super();
        //robot = new RobotLegoEV3();
        terminar = false;
        ligado = false;

    }

   /* public RobotLegoEV3 getRobot()
    {
```

```
118         return robot;
119     }*/
120 
121     public boolean getTerminar()
122     {
123         return terminar;
124     }
125 
126     public void setTerminar(boolean b)
127     {
128         terminar = b;
129     }
130 
131     public boolean isLigado()
132     {
133         return ligado;
134     }
135 
136     public void setLigado(boolean b)
137     {
138         ligado = b;
139     }
140 
141     public void setNome(String n)
142     {
143         nome = n;
144     }
145 
146     public String getNome()
147     {
148         return nome;
149     }
150 
151 }
152 
153 GUI_Subdito
154 
155 package ptrabalho;
156 
157 import java.awt.EventQueue;
158 
159 import javax.swing.BorderFactory;
160 import javax.swing.JFileChooser;
161 import javax.swing.JFrame;
162 import javax.swing.JPanel;
163 import javax.swing.border.Border;
164 import javax.swing.border.EmptyBorder;
165 import javax.swing.border.LineBorder;
166 import javax.swing.border.TitledBorder;
167 
168 import java.awt.Color;
169 import javax.swing.JLabel;
170 import java.awt.Font;
171 import java.awt.event.ActionEvent;
172 import java.awt.event.ActionListener;
173 import java.awt.event.WindowAdapter;
174 import java.awt.event.WindowEvent;
175 
176 import javax.swing.SwingConstants;
177 import javax.swing.JTextField;
```

```
178 import javax.swing.JRadioButton;
179 import javax.swing.JTextArea;
180
181 public class GUI_Subdito extends GUI_Base
182 {
183
184   private JPanel contentPane;
185   private JTextField txtNome;
186   private App_Subdito app;
187   private BD_Subdito bd;
188
189   /**
190    * Launch the application.
191    */
192   public GUI_Subdito(BD_Subdito bd)
193   {
194     super(bd);
195     EventQueue.invokeLater(new Runnable()
196     {
197       public void run()
198       {
199         try
200         {
201           init_GUI_Subdito(bd);
202
203         } catch (Exception e)
204         {
205           e.printStackTrace();
206         }
207       }
208     });
209   }
210
211   /**
212    * Create the frame.
213    */
214   public void init_GUI_Subdito(BD_Subdito bd)
215   {
216     setTitle("Trabalho 1 - GUI Subdito");
217     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
218     setBounds(100, 100, 754, 695);
219
220
221     JLabel lblNomeDoRobot = new JLabel("Nome do robot");
222     lblNomeDoRobot.setHorizontalAlignment(SwingConstants.LEFT);
223     lblNomeDoRobot.setFont(new Font("Arial", Font.BOLD, 12));
224     lblNomeDoRobot.setBounds(30, 61, 103, 25);
225     getContentPane().add(lblNomeDoRobot);
226
227
228     btnFrt.addActionListener(new ActionListener()
229     {
230       public void actionPerformed(ActionEvent e)
231       {
232         bd.getRobot().Reta(bd.getDist());
233             bd.getRobot().Parar(false);
234             txtLog.append(" O robot avançou " + bd.getDist() + "\n"
    );
235       }
236     });
```

```
237
238     btnEsq.addActionListener(new ActionListener()
239     {
240       public void actionPerformed(ActionEvent e)
241       {
242         bd.getRobot().CurvarEsquerda(bd.getRaio(),bd.getAng());
243             bd.getRobot().Parar(false);
244             txtLog.append(" O robot virou esquerda com raio = " +
    bd.getRaio() +" e angulo = " + bd.getAng() + "\n");
245       }
246     });
247
248     btnDir.addActionListener(new ActionListener()
249     {
250       public void actionPerformed(ActionEvent e)
251       {
252         bd.getRobot().CurvarDireita(bd.getRaio(),bd.getAng());
253             bd.getRobot().Parar(false);
254             txtLog.append(" O robot virou direita com raio = " + bd
    .getRaio() +" e angulo = " + bd.getAng() + "\n");
255       }
256     });
257
258     btnTras.addActionListener(new ActionListener()
259     {
260       public void actionPerformed(ActionEvent e)
261       {
262         bd.getRobot().Reta(-(bd.getDist()));
263             bd.getRobot().Parar(false);
264             txtLog.append(" O robot recuou " + bd.getDist() + "\n")
    ;
265       }
266     });
267
268     btnParar.addActionListener(new ActionListener()
269     {
270       public void actionPerformed(ActionEvent e)
271       {
272         bd.getRobot().Parar(true);
273         System.out.print(bd.getCanal().GetandSetReadLeitor());
274         txtLog.append(" O robot parou \n");
275         //txtLog.append("no limpa " + bd.getCanal().GetandSetRead()
    .toString() +"\n");
276
277       }
278     });
279
280
281     JRadioButton rdbtnAbrirFecharBlt = new JRadioButton("Abrir /
    Fechar Bluetooth");
282     rdbtnAbrirFecharBlt.addActionListener(new ActionListener()
283     {
284       public void actionPerformed(ActionEvent e) {
285         if (bd.isLigado())
286             {
287           System.out.println("Desligando...");
288                 bd.getRobot().CloseEV3();
289                 bd.setLigado(false);
290
291             } else
```

```
292             {
293               System.out.println("Ligando...");
294                 bd.setLigado(bd.getRobot().OpenEV3(bd.getNome()));
295                 System.out.println(bd.isLigado());
296                 bd.setLigado(true);
297             }
298       }
299     });
300
301     rdbtnAbrirCanal.addActionListener(new ActionListener()
302     {
303       public void actionPerformed(ActionEvent e)
304       {
305         if (rdbtnAbrirCanal.isSelected())
306         {
307           CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente(bd.getNMensagens());
308           cc.abrirCanalLeitor(bd.getFile());
309           bd.setCanal(cc);
310           txtLog.append(" Canal de Comunicação aberto!! \n");
311
312           //while(rdbtnAbrirCanal.isSelected())
313
314             }
315         else {
316           bd.getCanal().fecharCanal();
317           bd.setCanal(null);
318           txtLog.append(" Canal de Comunicação fechado \n");
319         }
320       }
321     });
322
323
324     txtNome = new JTextField();
325     txtNome.addActionListener(new ActionListener()
326     {
327       public void actionPerformed(ActionEvent e)
328       {
329         bd.setNome(txtNome.getText());
330         txtLog.append(" O nome do Robot é " + bd.getNome() + "\n");
331       }
332     });
333
334
335     txtNome.setFont(new Font("Arial", Font.BOLD, 12));
336     txtNome.setColumns(10);
337     txtNome.setBounds(143, 61, 279, 25);
338     getContentPane().add(txtNome);
339
340
341
342     rdbtnAbrirFecharBlt.setFont(new Font("Arial", Font.BOLD, 12));
343     rdbtnAbrirFecharBlt.setBounds(455, 61, 158, 25);
344     getContentPane().add(rdbtnAbrirFecharBlt);
345
346
347     Border bords_robot= BorderFactory.createLineBorder(new Color
    (0,0,0),1);
348     TitledBorder borda_robot = BorderFactory.createTitledBorder(
    bords_robot, "Robot");
```

```
    JPanel panel = new JPanel();
    panel.setName("Canal de Comunicação");
    panel.setBorder(new LineBorder(new Color(0, 0, 0)));
    panel.setBounds(10, 34, 719, 66);
    panel.setBorder(borda_robot);
    getContentPane().add(panel);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(100, 100,100, 100));

    addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                if (bd.isLigado())
                {
                    txtLog.append(" Desconectando o robot ... \n");
                    System.out.println("desconect");
                    bd.getRobot().CloseEV3();
                    rdbtnAbrirFecharBlt.setSelected(false);
                }
                else {
                    System.out.println("Closing program");
                    System.exit(0);
                }
            }
        });

    setVisible(true);
  }

}
```

Classe BD_Rei

```
package ptrabalho;


//import robot.RobotLegoEV3;

public class BD_Rei extends BD_Base
{
    //private RobotLegoEV3 robot;
    private boolean terminar;
    private boolean ligado;

    private String nome;


    public BD_Rei()
    {
      super();
        terminar = false;
        ligado = false;

    }


    public boolean getTerminar()
    {
        return terminar;
    }

```

```
409     public void setTerminar(boolean b)
410     {
411         terminar = b;
412     }
413 
414     public boolean isLigado()
415     {
416         return ligado;
417     }
418 
419     public void setLigado(boolean b)
420     {
421         ligado = b;
422     }
423 
424     public void setNome(String n)
425     {
426         nome = n;
427     }
428 
429     public String getNome()
430     {
431         return nome;
432     }
433 
434 }
435 
436 
437 Classe BD_Base
438 
439 package ptrabalho;
440 
441 import java.util.ArrayList;
442 import java.util.List;
443 
444 public class BD_Base {
445 
446   protected int distance;
447   protected int angulo;
448     protected int raio;
449     protected String file;
450     protected CanalComunicacaoConsistente ccc = null;
451     protected int nMensagens = 8;
452     private List<Mensagem> mensagens = new ArrayList<>();
453 
454   public BD_Base() {
455     distance = 30;
456         angulo = 90;
457         raio = 20;
458   }
459 
460   public int getDist()
461     {
462         return distance;
463     }
464 
465     public void setDist(int i)
466     {
467         distance = i;
468     }
```

```
469
470     public int getAng()
471     {
472         return angulo;
473     }
474
475     public void setAng(int i)
476     {
477         angulo = i;
478     }
479
480     public int getRaio()
481     {
482         return raio;
483     }
484
485     public void setRaio(int i)
486     {
487         raio = i;
488     }
489
490     public String getFile()
491     {
492         return file;
493     }
494
495     public void setFile(String f)
496     {
497         file = f;
498     }
499
500     public CanalComunicacaoConsistente getCanal()
501     {
502         return ccc;
503     }
504
505     public void setCanal(CanalComunicacaoConsistente c)
506     {
507         ccc = c;
508     }
509
510     public int getNMensagens()
511     {
512         return nMensagens;
513     }
514
515     public void setNMensagens(int i)
516     {
517       nMensagens = i;
518     }
519
520     public void addMensagem(Mensagem msg)
521     {
522       mensagens.add(msg);
523     }
524
525     public void removeMensagem()
526     {
527       mensagens.remove(0);
528     }
```

```
529
530     public List<Mensagem> getMensagens()
531     {
532     return mensagens;
533     }
534
535
536 }
537
538
539 Classe GUI_Rei
540
541 package ptrabalho;
542
543 import java.awt.Color;
544 import java.awt.EventQueue;
545 import java.awt.Font;
546 import java.awt.event.ActionEvent;
547 import java.awt.event.ActionListener;
548 import java.util.Random;
549
550 import javax.swing.BorderFactory;
551 import javax.swing.JButton;
552 import javax.swing.JFrame;
553 import javax.swing.JPanel;
554 import javax.swing.border.Border;
555 import javax.swing.border.EmptyBorder;
556 import javax.swing.border.LineBorder;
557 import javax.swing.border.TitledBorder;
558
559
560
561
562 public class GUI_Rei extends GUI_Base
563 {
564   private int id = 0;
565
566   private JPanel contentPane;
567
568   private int state = 0;
569   private final int gerarRandom = 0;
570   private final int escreverMensagem = 1;
571   private final int dormir = 2;
572   private final int esperarTempoExecucao = 3;
573
574   Mensagem msg = null;
575
576   /**
577    * Launch the application.
578    */
579   /*public static void main(String[] args)
580   {
581     EventQueue.invokeLater(new Runnable()
582     {
583       public void run() {
584         try
585         {
586           GUI_Rei frame = new GUI_Rei(bd);
587           frame.setVisible(true);
588         } catch (Exception e)
```

```
589         {
590           e.printStackTrace();
591         }
592       }
593     });
594   }*/
595 
596   public GUI_Rei(BD_Rei bd)
597   {
598     super(bd);
599     EventQueue.invokeLater(new Runnable()
600     {
601       public void run()
602       {
603         try
604         {
605           init_GUI_Rei(bd);
606 
607         } catch (Exception e)
608         {
609           e.printStackTrace();
610         }
611       }
612     });
613   }
614 
615   /**
616    * Create the frame.
617    */
618   public void init_GUI_Rei(BD_Rei bd)
619   {
620     setTitle("Trabalho 1 - GUI Rei");
621     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
622     setBounds(100, 100, 754, 697);
623 
624     Border cost = BorderFactory.createLineBorder(new Color(0,0,0)
    ,1);
625     TitledBorder borda_rei = BorderFactory.createTitledBorder(cost,
    "Controle do Robot em Modo Automático");
626     JPanel panel_1_1 = new JPanel();
627     panel_1_1.setLayout(null);
628     panel_1_1.setName("Controle do Robot em Modo Automático");
629     panel_1_1.setBorder(new LineBorder(new Color(0, 0, 0)));
630     panel_1_1.setBounds(10, 34, 719, 66);
631     panel_1_1.setBorder(borda_rei);
632     getContentPane().add(panel_1_1);
633 
634     JButton btn8com = new JButton("8 Comandos Aleatórios");
635     btn8com.setEnabled(false);
636     btn8com.setFont(new Font("Arial", Font.BOLD, 12));
637     btn8com.setBounds(10, 26, 300, 25);
638     panel_1_1.add(btn8com);
639 
640     JButton btn16com = new JButton("16 Comandos Aleatórios");
641     btn16com.setEnabled(false);
642     btn16com.setFont(new Font("Arial", Font.BOLD, 12));
643     btn16com.setBounds(385, 26, 300, 25);
644     panel_1_1.add(btn16com);
645     contentPane = new JPanel();
646     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
```

```
647
648     rdbtnAtivarDesativarComp.addActionListener(new ActionListener()
649     {
650       public void actionPerformed(ActionEvent e)
651       {
652         if(!rdbtnAtivarDesativarComp.isSelected())
653         {
654           btn8com.setEnabled(false);
655           btn16com.setEnabled(false);
656         }
657         else {
658           btn8com.setEnabled(true);
659           btn16com.setEnabled(true);
660         }
661       }
662     });
663
664     btnFrt.addActionListener(new ActionListener()
665     {
666       public void actionPerformed(ActionEvent e)
667       {
668         Mensagem mensagem = new Mensagem(id,1,bd.getDist(),0);
669         bd.addMensagem(mensagem);
670       }
671     });
672
673     btnEsq.addActionListener(new ActionListener()
674     {
675       public void actionPerformed(ActionEvent e)
676       {
677         Mensagem mensagem = new Mensagem(id,3,bd.getRaio(),bd.
    getAng());
678         bd.addMensagem(mensagem);
679       }
680     });
681
682     btnDir.addActionListener(new ActionListener()
683     {
684       public void actionPerformed(ActionEvent e)
685       {
686         Mensagem mensagem = new Mensagem(id,2,bd.getRaio(),bd.
    getAng());
687         bd.addMensagem(mensagem);
688       }
689     });
690
691     btnTras.addActionListener(new ActionListener()
692     {
693       public void actionPerformed(ActionEvent e)
694       {
695         Mensagem mensagem = new Mensagem(id,1,-bd.getDist(),0);
696         bd.addMensagem(mensagem);
697       }
698     });
699
700     btnParar.addActionListener(new ActionListener()
701     {
702       public void actionPerformed(ActionEvent e)
703       {
704         Mensagem mensagem = new Mensagem(id,0,0,0);
```

```
705         bd.addMensagem(mensagem);
706       }
707     });
708 
709     btn8com.addActionListener(new ActionListener()
710     {
711       public void actionPerformed(ActionEvent e)
712       {
713         for (int i = 0; i<8; i++)
714         {
715           msg = gerarRandomMensagem();
716           bd.addMensagem(msg);
717 
718         }
719       }
720     });
721 
722     btn16com.addActionListener(new ActionListener()
723     {
724       public void actionPerformed(ActionEvent e)
725       {
726         for (int i = 0; i<16; i++)
727         {
728           msg = gerarRandomMensagem();
729           bd.addMensagem(msg);
730         }
731 
732       }
733     });
734 
735     rdbtnAbrirCanal.addActionListener(new ActionListener()
736     {
737       public void actionPerformed(ActionEvent e)
738       {
739         if (rdbtnAbrirCanal.isSelected())
740         {
741           CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente(bd.getNMensagens());
742           cc.abrirCanalEscritor(bd.getFile());
743           bd.setCanal(cc);
744           txtLog.append(" Canal de Comunicação aberto!! \n");
745             }
746         else {
747           bd.getCanal().fecharCanal();
748           bd.setCanal(null);
749           txtLog.append(" Canal de Comunicação fechado \n");
750         }
751       }
752     });
753 
754 
755 
756     setVisible(true);
757   }
758 
759 
760 
761   private Mensagem gerarRandomMensagem() {
762     Mensagem m = new Mensagem();
763     Random rn = new Random();
```

```
764     int[] variaveis = new int[4]; // array de 4 variaveis
765     int tipoMensagem = rn.nextInt(3); // random between 0-2
766     if(id==8)
767       id=0;
768     variaveis = gerarVariaveis(tipoMensagem);
769     m.setId(variaveis[0]);
770     m.setTipo(variaveis[1]);
771     m.setArg1(variaveis[2]);
772     m.setArg2(variaveis[3]);
773     return m;
774   }
775 
776   /*
777    * Metodo auxiliar ao gerarRandomMensagem()
778    *
779    * @param tMsg - tipo de mensagem
780    */
781   private int[] gerarVariaveis(int tMsg) {
782     Random rn = new Random();
783     int[] variaveis = new int[4];
784     int variavel;
785     if (tMsg == 0) { // para tMsg 0 faz reta
786       variaveis[0] = id;
787       variaveis[1] = 1; // 1 equivale a reta na minha mensagem
788       variavel = rn.nextInt(45) + 5; // random between 5-50 cm reta
789       variaveis[2] = variavel;
790       variaveis[3] = 0;
791     } else if (tMsg == 1) { // para tMsg 1 faz curva direita
792       variaveis[0] = id;
793       variaveis[1] = 2; // 2 equivale a reta na minha mensagem
794       variavel = rn.nextInt(30); // random between 0-30 raio
795       variaveis[2] = variavel;
796       variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
797       variaveis[3] = variavel;
798     } else { // para tMsg 0 faz curva esquerda
799       variaveis[0] = id;
800       variaveis[1] = 3; // 3 equivale a reta na minha mensagem
801       variavel = rn.nextInt(30); // random between 0-30 raio
802       variaveis[2] = variavel;
803       variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
804       variaveis[3] = variavel;
805     }
806     return variaveis;
807   }
808 }
809 
810 
811 Classe GUI_Base
812 
813 package ptrabalho;
814 
815 import java.awt.EventQueue;
816 
817 import javax.swing.JFrame;
818 import javax.swing.JPanel;
819 import javax.swing.border.Border;
820 import javax.swing.border.EmptyBorder;
821 import javax.swing.BoxLayout;
822 import java.awt.GridLayout;
823 
```

```
824 import javax.swing.BorderFactory;
825 import javax.swing.Box;
826 import java.awt.CardLayout;
827 import javax.swing.JTextField;
828 import java.awt.FlowLayout;
829 import javax.swing.JLabel;
830 import javax.swing.SwingConstants;
831 import java.awt.Font;
832 import javax.swing.JButton;
833 import javax.swing.JToggleButton;
834 import javax.swing.JSpinner;
835 import javax.swing.JRadioButton;
836 import javax.swing.JSeparator;
837 import javax.swing.JComboBox;
838 import javax.swing.JFileChooser;
839 import javax.swing.JTextPane;
840 import java.awt.Panel;
841 import java.awt.Color;
842 import java.awt.Canvas;
843 import javax.swing.border.LineBorder;
844 import javax.swing.border.TitledBorder;
845 import javax.swing.event.ChangeEvent;
846 import javax.swing.event.ChangeListener;
847 import javax.swing.JCheckBox;
848 import java.awt.event.ActionListener;
849 import java.io.File;
850 import java.awt.event.ActionEvent;
851 import javax.swing.JTextArea;
852 import javax.swing.JScrollPane;
853 import javax.swing.SpinnerNumberModel;
854
855 public class GUI_Base extends JFrame
856 {
857
858   private JPanel contentPane;
859   protected JTextField txtRaio;
860   protected JTextField txtAng;
861   protected JTextField txtDist;
862   protected JTextField txtFile;
863   protected JButton btnFrt;
864   protected JButton btnEsq;
865   protected JButton btnDir;
866   protected JButton btnParar;
867   protected JButton btnTras;
868   protected JTextArea txtLog;
869   protected JButton btnFile;
870   protected JButton btnLimpaLog;
871   protected JCheckBox rdbtnAtivarDesativarComp;
872   protected JRadioButton rdbtnAbrirCanal;
873   protected JSpinner spinNMensagem;
874
875
876   /**
877    * Launch the application.
878    */
879   /*public static void main(String[] args)
880   {
881     EventQueue.invokeLater(new Runnable()
882     {
883       public void run()
```

```
884       {
885         try
886         {
887           GUI_Base frame = new GUI_Base();
888           frame.setVisible(true);
889         } catch (Exception e)
890         {
891           e.printStackTrace();
892         }
893       }
894     });
895   }*/
896
897   /**
898    * Create the frame.
899    */
900   public GUI_Base(BD_Base bd)
901   {
902     setTitle("Trabalho 1 - GUI Base");
903     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
904     setBounds(100, 100, 754, 691);
905     contentPane = new JPanel();
906     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
907
908     setContentPane(contentPane);
909     contentPane.setLayout(null);
910
911     txtFile = new JTextField();
912     txtFile.setFont(new Font("Arial", Font.BOLD, 12));
913     txtFile.setBounds(143, 156, 494, 25);
914     contentPane.add(txtFile);
915     txtFile.setColumns(10);
916
917     JLabel lblNewLabel = new JLabel("Ficheiro do Canal");
918     lblNewLabel.setFont(new Font("Arial", Font.BOLD, 12));
919     lblNewLabel.setHorizontalAlignment(SwingConstants.LEFT);
920     lblNewLabel.setBounds(30, 156, 103, 25);
921     contentPane.add(lblNewLabel);
922
923     btnFile = new JButton("...");
924     btnFile.setFont(new Font("Arial", Font.BOLD, 12));
925     btnFile.setBounds(647, 157, 61, 25);
926     contentPane.add(btnFile);
927
928     JLabel lblNMsg = new JLabel("N   msg");
929     lblNMsg.setHorizontalAlignment(SwingConstants.LEFT);
930     lblNMsg.setFont(new Font("Arial", Font.BOLD, 12));
931     lblNMsg.setBounds(30, 200, 103, 25);
932     contentPane.add(lblNMsg);
933
934     spinNMensagem = new JSpinner();
935     spinNMensagem.setModel(new SpinnerNumberModel(8, 8, 12, 1));
936     spinNMensagem.setFont(new Font("Arial", Font.BOLD, 12));
937     spinNMensagem.setBounds(143, 200, 50, 25);
938     contentPane.add(spinNMensagem);
939
940     rdbtnAbrirCanal = new JRadioButton("Abrir/Fechar Canal");
941     rdbtnAbrirCanal.setFont(new Font("Arial", Font.BOLD, 12));
942     rdbtnAbrirCanal.setBounds(459, 200, 103, 25);
943     contentPane.add(rdbtnAbrirCanal);
```

```
944
945     Border simpb = BorderFactory.createLineBorder(new Color(0,0,0)
        ,1);
946     TitledBorder border_simp = BorderFactory.createTitledBorder(
        simpb,"Canal de Comunicação");
947     JPanel panel = new JPanel();
948     panel.setName("Canal de Comunicação");
949     panel.setBorder(new LineBorder(new Color(0, 0, 0)));
950     panel.setBounds(10, 128, 719, 113);
951     panel.setBorder(border_simp);
952     contentPane.add(panel);
953
954     JLabel lblRaio = new JLabel("Raio");
955     lblRaio.setHorizontalAlignment(SwingConstants.LEFT);
956     lblRaio.setFont(new Font("Arial", Font.BOLD, 12));
957     lblRaio.setBounds(30, 270, 103, 25);
958     contentPane.add(lblRaio);
959
960     txtRaio = new JTextField();
961     txtRaio.setText("20");
962
963     txtRaio.setFont(new Font("Arial", Font.BOLD, 12));
964     txtRaio.setColumns(10);
965     txtRaio.setBounds(143, 270, 100, 23);
966     contentPane.add(txtRaio);
967
968     JLabel lblngulo = new JLabel("Ângulo");
969     lblngulo.setHorizontalAlignment(SwingConstants.LEFT);
970     lblngulo.setFont(new Font("Arial", Font.BOLD, 12));
971     lblngulo.setBounds(30, 300, 103, 25);
972     contentPane.add(lblngulo);
973
974     JLabel lblDistncia = new JLabel("Distância");
975     lblDistncia.setHorizontalAlignment(SwingConstants.LEFT);
976     lblDistncia.setFont(new Font("Arial", Font.BOLD, 12));
977     lblDistncia.setBounds(30, 330, 103, 25);
978     contentPane.add(lblDistncia);
979
980     txtAng = new JTextField();
981     txtAng.setText("90");
982     txtAng.setFont(new Font("Arial", Font.BOLD, 12));
983     txtAng.setColumns(10);
984     txtAng.setBounds(143, 300, 100, 23);
985     contentPane.add(txtAng);
986
987     txtDist = new JTextField();
988     txtDist.setText("30");
989     txtDist.setFont(new Font("Arial", Font.BOLD, 12));
990     txtDist.setColumns(10);
991     txtDist.setBounds(143, 330, 100, 23);
992     contentPane.add(txtDist);
993
994     btnParar = new JButton("Parar");
995     btnParar.setEnabled(false);
996     btnParar.setFont(new Font("Arial", Font.BOLD, 12));
997     btnParar.setBounds(470, 297, 100, 25);
998     contentPane.add(btnParar);
999
1000    btnFrt = new JButton("Frente");
1001    btnFrt.setEnabled(false);
```

```
1002     btnFrt.setFont(new Font("Arial", Font.BOLD, 12));
1003     btnFrt.setBounds(470, 269, 100, 25);
1004     contentPane.add(btnFrt);
1005 
1006     btnDir = new JButton("Direita");
1007     btnDir.setEnabled(false);
1008     btnDir.setFont(new Font("Arial", Font.BOLD, 12));
1009     btnDir.setBounds(575, 298, 100, 25);
1010     contentPane.add(btnDir);
1011 
1012     btnEsq = new JButton("Esquerda");
1013     btnEsq.setEnabled(false);
1014     btnEsq.setFont(new Font("Arial", Font.BOLD, 12));
1015     btnEsq.setBounds(365, 297, 100, 25);
1016     contentPane.add(btnEsq);
1017 
1018     btnTras = new JButton("Tras");
1019     btnTras.setEnabled(false);
1020     btnTras.setFont(new Font("Arial", Font.BOLD, 12));
1021     btnTras.setBounds(470, 325, 100, 25);
1022     contentPane.add(btnTras);
1023 
1024     Border borda_cont_robot = BorderFactory.createLineBorder(new
    Color(0,0,0),1);
1025     TitledBorder borda1 = BorderFactory.createTitledBorder(
    borda_cont_robot,"Controle do Robot");
1026     JPanel panel_1 = new JPanel();
1027     panel_1.setFont(new Font("Arial", Font.BOLD, 12));
1028     panel_1.setName("Controle do Robot");
1029     panel_1.setBorder(new LineBorder(new Color(0, 0, 0)));
1030     panel_1.setBounds(10, 251, 719, 113);
1031     panel_1.setBorder(borda1);
1032     contentPane.add(panel_1);
1033 
1034     rdbtnAtivarDesativarComp = new JCheckBox("Ativar / Desativar
    Comportamento");
1035     rdbtnAtivarDesativarComp.setFont(new Font("Arial", Font.BOLD,
    12));
1036     rdbtnAtivarDesativarComp.setBounds(10, 365, 262, 45);
1037     contentPane.add(rdbtnAtivarDesativarComp);
1038 
1039     JLabel lblLog = new JLabel("Log");
1040     lblLog.setHorizontalAlignment(SwingConstants.LEFT);
1041     lblLog.setFont(new Font("Arial", Font.BOLD, 12));
1042     lblLog.setBounds(10, 429, 103, 25);
1043     contentPane.add(lblLog);
1044 
1045     btnLimpaLog = new JButton("Limpar Log");
1046     btnLimpaLog.setFont(new Font("Arial", Font.BOLD, 12));
1047     btnLimpaLog.setBounds(10, 630, 719, 25);
1048     contentPane.add(btnLimpaLog);
1049 
1050 
1051     txtLog = new JTextArea();
1052     txtLog.setBounds(45, 464, 659, 156);
1053     contentPane.add(txtLog);
1054 
1055     JScrollPane scrollPane = new JScrollPane(txtLog);
1056     scrollPane.setBounds(45, 464, 663, 156);
1057     contentPane.add(scrollPane);
```

```
1058
1059     btnLimpaLog.addActionListener(new ActionListener()
1060     {
1061       public void actionPerformed(ActionEvent e)
1062       {
1063         txtLog.setText("");
1064       }
1065     });
1066
1067     spinNMensagem.addChangeListener(new ChangeListener() {
1068       @Override
1069       public void stateChanged(ChangeEvent e) {
1070         int nMensagens = (int) spinNMensagem.getValue();
1071         if (bd.getCanal()!= null) {
1072
1073           bd.getCanal().nMensagens = nMensagens;
1074           bd.getCanal().BUFFER_MAX = 16 * nMensagens;
1075
1076         }
1077         bd.setNMensagens(nMensagens);
1078         txtLog.append(" O num de mensagens é " + nMensagens + "\n")
     ;
1079       }
1080     });
1081
1082     btnFile.addActionListener(new ActionListener()
1083     {
1084       public void actionPerformed(ActionEvent e)
1085       {
1086         txtLog.append(" Escolher Ficheiro \n");
1087         JFileChooser fileChooser = new JFileChooser(System.
     getProperty("user.dir"));
1088
1089         if (fileChooser.showSaveDialog(null) == JFileChooser.
     APPROVE_OPTION) {
1090           String file = fileChooser.getSelectedFile().
     getAbsolutePath();
1091           txtFile.setText(file);
1092           bd.setFile(file);
1093           txtLog.append(" O nome do ficheiro é " + bd.getFile() + "
     \n");
1094         }
1095       }
1096     });
1097
1098     txtFile.addActionListener(new ActionListener()
1099     {
1100       public void actionPerformed(ActionEvent e)
1101       {
1102         bd.setFile(txtFile.getText());
1103         txtLog.append(" O nome do ficheiro é " + bd.getFile() + "\n
     ");
1104       }
1105     });
1106
1107
1108     rdbtnAtivarDesativarComp.addActionListener(new ActionListener()
1109     {
1110       public void actionPerformed(ActionEvent e)
1111       {
```

```
1112          if(!rdbtnAtivarDesativarComp.isSelected())
1113          {
1114            btnFrt.setEnabled(false);
1115            btnEsq.setEnabled(false);
1116            btnParar.setEnabled(false);
1117            btnDir.setEnabled(false);
1118            btnTras.setEnabled(false);
1119          }
1120          else {
1121            btnFrt.setEnabled(true);
1122            btnEsq.setEnabled(true);
1123            btnParar.setEnabled(true);
1124            btnDir.setEnabled(true);
1125            btnTras.setEnabled(true);
1126          }
1127        }
1128      });
1129
1130
1131
1132
1133      txtDist.addActionListener(new ActionListener()
1134      {
1135        public void actionPerformed(ActionEvent e)
1136        {
1137          bd.setDist(Integer.parseInt(txtDist.getText()));
1138          txtLog.append(" A distancia é " + bd.getDist() + "\n");
1139        }
1140      });
1141
1142      txtAng.addActionListener(new ActionListener()
1143      {
1144        public void actionPerformed(ActionEvent e)
1145        {
1146          bd.setAng(Integer.parseInt(txtAng.getText()));
1147          txtLog.append(" O angulo é " + bd.getAng() + "\n");
1148        }
1149      });
1150
1151      txtRaio.addActionListener(new ActionListener()
1152      {
1153        public void actionPerformed(ActionEvent e)
1154        {
1155          bd.setRaio(Integer.parseInt(txtRaio.getText()));
1156          txtLog.append(" A raio é " + bd.getRaio() + "\n");
1157        }
1158      });
1159
1160    }
1161  }
1162
1163  Classe CanalComunicacaoConsistente
1164
1165  package ptrabalho;
1166
1167  public class CanalComunicacaoConsistente extends CanalComunicacao{
1168
1169
1170      public CanalComunicacaoConsistente(int n) {
1171      super(n);
```

```
1172      // TODO Auto-generated constructor stub
1173    }
1174
1175    public boolean GetandSetWrite(Mensagem msg) {
1176          Mensagem mensagem = GetandSetRead();
1177          //System.out.println(mensagem);
1178          try {
1179            if(mensagem == null)
1180              return false;
1181              if(mensagem.tipo == iMensagem.vazia) {
1182                  System.out.println("entrou, no buffer: " + mensagem
      + "enviei msg = " + msg);
1183
1184                  fl = canal.lock();
1185
1186                  enviarMensagem(msg, true);
1187
1188                  fl.release();
1189                  return true;
1190              }else {
1191                //System.out.println("NÃO entrou, no buffer: " +
       mensagem + "quero enviar msg = " + msg);
1192              }
1193          } catch (Exception e) {
1194              e.printStackTrace();
1195          }
1196
1197          return false;
1198      }
1199
1200      public Mensagem GetandSetRead(){
1201
1202          try {
1203              fl = canal.lock();
1204
1205              Mensagem mensagem = receberMensagem(true);
1206
1207              //limparLida();
1208
1209              fl.release();
1210
1211              return mensagem;
1212
1213          } catch (Exception e) {
1214              e.printStackTrace();
1215          }
1216
1217          return null;
1218      }
1219
1220      public Mensagem GetandSetReadLeitor(){
1221
1222          try {
1223              fl = canal.lock();
1224
1225              Mensagem mensagem = receberMensagem(false);
1226              System.out.println(mensagem);
1227
1228              limparLida();
1229
```

```
            fl.release();

            return mensagem;

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

        return null;
    }



    /*public static void main (String[] args) {
        CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente();

        MensagemParar msgParar = new MensagemParar(0, 0, false);
    //int id, int tipo, boolean sincrono
        MensagemReta msg     = new MensagemReta(1,1,20);         //
    int id, int tipo, int dist
        MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15);
    //int id, int tipo, int raio, int ang
        MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);

        //Testar
        cc.abrirCanal("comunicacao.dat");


        cc.GetandSetWrite(msg);
        System.out.println(cc.receberMensagem());
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCE);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgParar);
        cc.GetandSetWrite(msgParar);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgParar);
        cc.GetandSetWrite(msgParar);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgCD);
        cc.GetandSetWrite(msgParar);
        cc.GetandSetWrite(msgParar);

        cc.fecharCanal();
    }*/

}

Classe CanalComunicacao

package ptrabalho;
import java.io.*;
import java.nio.MappedByteBuffer;
```

```
1286 import java.nio.channels.FileChannel;
1287 import java.nio.channels.FileLock;
1288
1289 public class CanalComunicacao {
1290     FileLock fl;
1291
1292     // ficheiro
1293     private File ficheiro;
1294
1295     // canal que liga o conteúdo do ficheiro ao Buffer
1296     protected FileChannel canal;
1297
1298     // buffer
1299     private MappedByteBuffer buffer;
1300
1301     // dimensão máxima em bytes do buffer = (N   grup + 1) * 4
     bytes (int) * 4 ints
1302     //final int nGrupo = 15;
1303     //final int BUFFER_MAX = (nGrupo + 1) * 4 * 4;
1304     protected int nMensagens = 8;
1305     protected int BUFFER_MAX = nMensagens*16;
1306     protected int putBuffer, getBuffer;
1307
1308
1309
1310     //Construtor onde se cria o canal
1311     public CanalComunicacao(int n) {
1312         ficheiro = null;
1313         canal    = null;
1314         buffer   = null;
1315         nMensagens = n;
1316         BUFFER_MAX = nMensagens*16;
1317     }
1318
1319     public boolean abrirCanalEscritor(String Filename) {
1320         // cria um ficheiro
1321         ficheiro = new File(Filename);
1322
1323         //cria um canal de comunicação de leitura e escrita
1324         try {
1325             canal = new RandomAccessFile(ficheiro, "rw").getChannel
     ();
1326         } catch (FileNotFoundException e) {
1327             return false;
1328         }
1329
1330         // mapeia para memória o conteúdo do ficheiro
1331         try {
1332             buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
     BUFFER_MAX);
1333
1334             //Inicializar apontadores - put e get buffer a zero
1335             getBuffer = 0;
1336             putBuffer = 0;
1337
1338             //Inicializar buffer - dizer que todos os blocos são
     vazios
1339             inicializarBuffer();
1340
1341
```

```
1342        } catch (IOException e) {
1343            return false;
1344        }
1345
1346        return true;
1347    }
1348
1349    public boolean abrirCanalLeitor(String Filename) {
1350        // cria um ficheiro
1351        ficheiro = new File(Filename);
1352
1353        //cria um canal de comunicação de leitura e escrita
1354        try {
1355            canal = new RandomAccessFile(ficheiro, "rw").getChannel
    ();
1356        } catch (FileNotFoundException e) {
1357            return false;
1358        }
1359
1360        // mapeia para memória o conteúdo do ficheiro
1361        try {
1362            buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
    BUFFER_MAX);
1363
1364            //Inicializar apontadores - put e get buffer a zero
1365            getBuffer = 0;
1366            putBuffer = 0;
1367
1368
1369
1370
1371        } catch (IOException e) {
1372            return false;
1373        }
1374
1375        return true;
1376    }
1377
1378    // recebe e converte numa Mensagem, lê e retorna mensagem
1379    //true escritor// false leitor
1380    Mensagem receberMensagem(boolean e_l) {
1381
1382        int id, tipo, distancia, raio, angulo;
1383        boolean sincrono;
1384
1385        buffer.asIntBuffer();
1386        //buffer.position(0);
1387        //System.out.println("getbuffer" + getBuffer + "nMensanges"
    + nMensagens);
1388        buffer.position(getBuffer * 16);
1389
1390        id   = buffer.getInt();
1391        tipo = buffer.getInt();
1392
1393        Mensagem msg = null;
1394
1395        switch (tipo) {
1396            case iMensagem.parar:
1397                sincrono = buffer.getInt() == 1;
1398                msg = new MensagemParar(id, tipo, sincrono);
```

```
1399                 break;
1400
1401             case iMensagem.reta:
1402                 distancia = buffer.getInt();
1403                 msg = new MensagemReta(id, tipo, distancia);
1404                 break;
1405
1406             case iMensagem.curvarDir:
1407                 raio = buffer.getInt();
1408                 angulo = buffer.getInt();
1409                 msg = new MensagemCurvar(id, tipo, raio, angulo);
1410                 break;
1411
1412             case iMensagem.curvarEsq:
1413                 raio = buffer.getInt();
1414                 angulo = buffer.getInt();
1415                 msg = new MensagemCurvar(id, tipo, raio, angulo);
1416                 break;
1417
1418             case iMensagem.vazia:
1419                 msg = new Mensagem(id, tipo, 0, 0);
1420                 break;
1421         }
1422
1423         if(e_l) {
1424           if(tipo==4)
1425             getBuffer =++ getBuffer % nMensagens;
1426         }
1427         else {
1428           if (tipo != 4)
1429               getBuffer =++ getBuffer % nMensagens;
1430         }
1431
1432         return msg;
1433
1434     }
1435
1436 // recebe e converte numa Mensagem, lê e retorna mensagem
1437     Mensagem receberMensagemLeitor() {
1438
1439         int id, tipo, distancia, raio, angulo;
1440         boolean sincrono;
1441
1442         buffer.asIntBuffer();
1443         //buffer.position(0);
1444         //System.out.println("getbuffer" + getBuffer + "nMensanges"
    + nMensagens);
1445         buffer.position(getBuffer * 16);
1446
1447         id   = buffer.getInt();
1448         tipo = buffer.getInt();
1449
1450         Mensagem msg = null;
1451
1452         switch (tipo) {
1453             case iMensagem.parar:
1454                 sincrono = buffer.getInt() == 1;
1455                 msg = new MensagemParar(id, tipo, sincrono);
1456                 break;
1457
```

```
1458            case iMensagem.reta:
1459                distancia = buffer.getInt();
1460                msg = new MensagemReta(id, tipo, distancia);
1461                break;
1462
1463            case iMensagem.curvarDir:
1464                raio = buffer.getInt();
1465                angulo = buffer.getInt();
1466                msg = new MensagemCurvar(id, tipo, raio, angulo);
1467                break;
1468
1469            case iMensagem.curvarEsq:
1470                raio = buffer.getInt();
1471                angulo = buffer.getInt();
1472                msg = new MensagemCurvar(id, tipo, raio, angulo);
1473                break;
1474
1475            case iMensagem.vazia:
1476                msg = new Mensagem(id, tipo, 0, 0);
1477                break;
1478        }
1479
1480        if (tipo != 4)
1481          getBuffer =++ getBuffer % nMensagens;
1482
1483        return msg;
1484
1485    }
1486
1487    // envia uma Mensagem como um conjunto de ints
1488    void enviarMensagem(Mensagem msg, boolean e_c) {
1489
1490        try {
1491            //buffer.position(0);
1492            buffer.position(putBuffer * 16);
1493
1494            // Obter ID e escrevê-lo no buffer
1495            int id = msg.getId();
1496            buffer.putInt(id);
1497
1498            // Obter Tipo e escrevê-lo no buffer
1499            int tipo = msg.getTipo();
1500            buffer.putInt(tipo);
1501
1502            // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
1503            switch (tipo) {
1504                case iMensagem.parar:
1505                    buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
1506                    break;
1507
1508                case iMensagem.reta:
1509                    buffer.putInt(msg.getArg1());
1510                    break;
1511
1512                case iMensagem.curvarDir:
1513                  System.out.println("yo");
1514                    buffer.putInt(msg.getArg1());
1515                    buffer.putInt(msg.getArg2());
```

```
1516                    break;
1517
1518                case iMensagem.curvarEsq:
1519                    buffer.putInt(msg.getArg1());
1520                    buffer.putInt(msg.getArg2());
1521                    break;
1522
1523                case iMensagem.vazia:
1524                    buffer.putInt(0);
1525                    System.out.println("limpei");
1526                    //id = -1;
1527                    break;
1528            }
1529
1530            if (e_c) {
1531              if (tipo != 4)
1532                  putBuffer =++ putBuffer % nMensagens;
1533            }
1534            else
1535            {
1536              putBuffer =++ putBuffer % nMensagens;
1537                msg.setId(id + 1);
1538            }
1539
1540            //msg.setId(id + 1);
1541
1542        }catch(Exception e) {
1543            e.printStackTrace();
1544        }
1545
1546    }
1547
1548    void enviarMensagemClean(Mensagem msg) {
1549
1550        try {
1551            //buffer.position(0);
1552            buffer.position(putBuffer * 16);
1553
1554            // Obter ID e escrevê-lo no buffer
1555            int id = msg.getId();
1556            buffer.putInt(id);
1557
1558            // Obter Tipo e escrevê-lo no buffer
1559            int tipo = msg.getTipo();
1560            buffer.putInt(tipo);
1561
1562            // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
1563            switch (tipo) {
1564                case iMensagem.parar:
1565                    buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
1566                    break;
1567
1568                case iMensagem.reta:
1569                    buffer.putInt(msg.getArg1());
1570                    break;
1571
1572                case iMensagem.curvarDir:
1573                  System.out.println("yo");
```

```
1574                    buffer.putInt(msg.getArg1());
1575                    buffer.putInt(msg.getArg2());
1576                    break;
1577
1578                case iMensagem.curvarEsq:
1579                    buffer.putInt(msg.getArg1());
1580                    buffer.putInt(msg.getArg2());
1581                    break;
1582
1583                case iMensagem.vazia:
1584                    buffer.putInt(0);
1585                    System.out.println("limpei");
1586                    //id = -1;
1587                    break;
1588            }
1589
1590            putBuffer =++ putBuffer % nMensagens;
1591            msg.setId(id + 1);
1592
1593        }catch(Exception e) {
1594            e.printStackTrace();
1595        }
1596
1597    }
1598
1599    // fecha o canal entre o buffer e o ficheiro
1600    void fecharCanal() {
1601        if (canal != null)
1602            try {
1603                canal.close();
1604            } catch (IOException e) {
1605                canal = null;
1606            }
1607    }
1608
1609    /**Colocar todos os slots com mensagens vazias*/
1610    //Mts duvidas, gestao do ID, so tipo e id
1611    void inicializarBuffer() {
1612
1613        //1 msg - 4 * 4 = 16 bytes
1614        MensagemVazia msg = new MensagemVazia(0, iMensagem.vazia);
1615        System.out.print(BUFFER_MAX);
1616
1617        for(int i = 0; i < nMensagens; i++) {
1618            enviarMensagem(msg, false);
1619        }
1620        buffer.clear();
1621        putBuffer=0;
1622        getBuffer=0;
1623
1624    }
1625
1626    void limparLida(){
1627      System.out.println(getBuffer);
1628      if(getBuffer != 0)
1629        buffer.position((getBuffer-1) * 16);
1630      else
1631        buffer.position((7) * 16);
1632        buffer.putInt(getBuffer);
1633        buffer.putInt(iMensagem.vazia);
```

```
1634
1635     }
1636
1637
1638     /*public static void main(String[] args) {
1639
1640
1641       MensagemParar msgParar = new MensagemParar(0, 0, false);   //
    int id, int tipo, boolean sincrono
1642         MensagemReta msg     = new MensagemReta(1,1,20);          //
    int id, int tipo, int dist
1643         MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15);
    //int id, int tipo, int raio, int ang
1644         MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);
1645         MensagemVazia msgV = new MensagemVazia(4,4);
1646
1647
1648         CanalComunicacao cc = new CanalComunicacao(8);
1649         cc.abrirCanalEscritor("comunicacao.dat");
1650
1651         cc.enviarMensagem(msg);
1652         Mensagem msg1 =cc.receberMensagemLeitor();
1653         System.out.println(msg1);
1654         cc.enviarMensagem(msgV);
1655         System.out.println(cc.receberMensagemLeitor());
1656         cc.enviarMensagem(msgCD);
1657         System.out.println(cc.receberMensagemLeitor());
1658         cc.enviarMensagem(msgCE);
1659         System.out.println(cc.receberMensagemLeitor());
1660
1661         cc.fecharCanal();
1662         }*/
1663 }
1664
1665 Classe iMensagem
1666
1667 package ptrabalho;
1668
1669 public interface iMensagem {
1670
1671     int parar          = 0;
1672     int reta           = 1;
1673     int curvarDir        = 2;
1674     int curvarEsq        = 3;
1675     int vazia             = 4;
1676     int pedir            = 5;
1677     int sensor            = 6;
1678     int workflow        = 7;
1679     int endWorkflow    = 8;
1680 }
1681
1682 Classe Mensagem
1683
1684 package ptrabalho;
1685
1686 public class Mensagem implements iMensagem{
1687     int tipo, id, arg1, arg2;
1688
1689     public Mensagem(int id, int tipo, int arg1, int arg2) {
1690         this.tipo = tipo;
```

```
1691        this.id = id;
1692        this.arg1 = arg1;
1693        this.arg2 = arg2;
1694    }
1695
1696    public Mensagem() {
1697
1698    }
1699
1700
1701    @Override
1702    public String toString() {
1703        return "Mensagem{" +
1704                " id= " + id +
1705                " tipo=" + tipo +
1706                " arg1=" + arg1 +
1707                " arg2=" + arg2 +
1708                '}';
1709    }
1710
1711    public int getTipo() {
1712        return tipo;
1713    }
1714
1715    public void setTipo(int tipo) {
1716        this.tipo = tipo;
1717    }
1718
1719    public int getId() {
1720        return id;
1721    }
1722
1723    public void setId(int id) {
1724        this.id = id;
1725    }
1726
1727    public int getArg1() {
1728        return arg1;
1729    }
1730
1731    public void setArg1(int arg) {
1732        this.arg1 = arg;
1733    }
1734
1735    public int getArg2() {
1736        return arg2;
1737    }
1738
1739    public void setArg2(int arg) {
1740        this.arg2 = id;
1741    }
1742
1743    public boolean equals(Mensagem mensagem) {
1744
1745        if(mensagem==null) return false;
1746
1747        return (this.getId() == mensagem.getId());
1748    }
1749 }
1750
```

Classe MensagemCurvar

```
package ptrabalho;

public class MensagemCurvar extends Mensagem{
  int raio, ang;

  public MensagemCurvar(int id, int tipo, int raio, int ang) {
    super(id, tipo, raio, ang);
    this.raio = raio;
    this.ang  = ang;
  }

  public int getRaio() {
    return raio;
  }

  public void setRaio(int raio) {
    this.raio = raio;
  }

  public int getAng() {
    return ang;
  }

  public void setAng(int ang) {
    this.ang = ang;
  }
}
```

Classe MensagemParar

```
package ptrabalho;

public class MensagemParar extends Mensagem{

  boolean sincrono;

  public MensagemParar(int id, int tipo, boolean sincrono) {
    super(id, tipo, 0, 0);
    this.sincrono = sincrono;
  }

  @Override
  public String toString() {
    return super.toString() +" " +
        "sincrono=" + sincrono +
        '}';
  }

  public boolean isSincrono() {
    return sincrono;
  }

  public void setSincrono(boolean sincrono) {
    this.sincrono = sincrono;
  }

}
```

```
Classe MensagemReta

package ptrabalho;

public class MensagemReta extends Mensagem{
  int dist;

  public MensagemReta(int id, int tipo, int dist) {
    super(id, tipo, dist, 0);
    this.dist = dist;
  }

  public MensagemReta(int id, int tipo, int dist, int arg2) {
    super(id, tipo, dist, 0);
    this.dist = dist;
  }

  public int getDist() {
    return dist;
  }

  public void setDist(int dist) {
    this.dist = dist;
  }

  @Override
  public String toString() {
    return super.toString() + " distancia= " + dist + "}";
  }
}

Classe MensagemVazia

package ptrabalho;

public class MensagemVazia extends Mensagem{
    public MensagemVazia(int id, int tipo) {
        super(id, tipo, 0, 0);
    }
}
```