Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos - 2324SI

1º 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

1 Introdução

Nesta quarta aula foi realizada a implementação de variadas classes responsáveis pela comunicação que será posteriormente estabelecida entre o GUI_Subdito e o GUI_Rei. As classes CanalComunicacao e CanalComunicacaoConsistente vão tratar de tudo o que envolver o canal de comunicação onde as mensagens serão passadas e a Consistente, em particular, irá garantir que as mensagens não se sobrepõem quando estiverem a ser enviadas instruções para o robot do Súbdito e do Rei.

A classe Mensagem será a base de todas as outras mensagens. Será criada como uma template para as outras classes a poderem extender, definindo assim o tipo de mensagens que podem ser enviadas.

2 Desenvolvimento

2.1 Classes Mensagem e Interface

Começamos pela criação da interface iMensagem. Esta tem o simples propósito de atribuir um valor a cada estado. Isto será relevante nos próximos pontos.

```
package ptrabalho;
public interface iMensagem {

    final byte typeParar    = 0;
    final byte typeReta     = 1;
    final byte typeDireita  = 2;
    final byte typeEsquerda = 3;
    final int typeVazio     = 4;
}
```

Criamos então a classe Mensagem e implementamos a interface previamente desenvolvida. O construtor desta passa 4 argumentos: **id**, **type**, **arg1** e **arg2**.

```
public Mensagem(int id, int type, int arg1, int arg2) {
    this.type = type;
    this.id = id;
    this.arg1 = arg1;
    this.arg2 = arg2;
}
```

id vai guardar o valor de cada mensagem correspondente à sua posição no buffer, **type** será o valor que foi definido na interface que corresponde aos diferentes estados possíveis, **arg1** será utilizado apenas por algumas mensagens (Ex: typeReta para definir a distância) assim como **arg2** (Ex: typeDireita ou typeEsquerda para raio e ângulo). Quando utilizados, **arg1** e **arg2** são 0. São criados Getters e Setters para cada variável nessa mesma classe.

2.2 Canal de Comunicação e Consistente

É criado uma classe CanalDeComunicacao (CC) e CanalDeComunicacaoConsistente (CCC) que têm como base uma template que foi fornecida pelos docentes. Na classe CC é criada uma variável buffer e os seus limites (dimensão das mensagens, quantidade de mensagens que o buffer consegue ler) para garantir um flow de mensagens pelo próprio CC. São criados métodos que retornam booleans tais como abrirCanalEscritor() e abrirCanalLeitor() que fazem a distinção entre quem envia (Rei) e quem recebe (Súbdito), maioritariamente pelo facto de que o Rei quer abrir o canal totalmente limpo. Foram criados métodos enviarMensagem() e receberMensagem() que, se tiverem espaço no buffer, vão buscar o textbfid e o **type**, retorna uma mensagem (enviarMensagem) e lê a informação da mensagem através de getters criados na classe Mensagem (receberMensagem).

São criados adicionalmente os métodos limparLida() que limpa as mensagens anteriores depois de lidas, inicializarBuffer() que simplesmente inicializa o buffer e fecharCanal() que fecha o canal de comunicação.

Após testado, ficados com os seguintes resultados:

```
getbuffer: 0, nMensanges: 8
Mensagem{ id= 1 tipo=1 arg1=20 arg2=0} distancia= 20}
limpei
getbuffer: 1, nMensanges: 8
Mensagem{ id= 4 tipo=4 arg1=0 arg2=0}
getbuffer: 1, nMensanges: 8
Mensagem{ id= 2 tipo=2 arg1=15 arg2=15}
getbuffer: 2, nMensanges: 8
Mensagem{ id= 3 tipo=3 arg1=12 arg2=30}
```

Podemos observar que o valor do buffer e o **id** ambos aumentam ao mesmo ritmo até chegar uma mensagem vazia que é escrita mas não incrementa o valor no buffer pois não chega a ser lida.

```
void enviarMensagem(Mensagem msg) {

    try {
        //buffer.position(0);
        buffer.position(putBuffer * 16);

        // Obter ID e escrevê-lo no buffer
        int id = msg.getId();
        buffer.putInt(id);

        // Obter Tipo e escrevê-lo no buffer
        int tipo = msg.getTipo();
        buffer.putInt(tipo);

        // Escrever o conteúdo no buffer, consoante o tipo da mensagem (Ex: para o tipo reta -> escrever distância
        switch (tipo) {
            case iMensagem.parar:
                buffer.putInt(((MensagemParar) msg).isSincrono() ? 1 : 0);
                break;

            case iMensagem.reta:
                buffer.putInt(msg.getArg1());
                break;

            case iMensagem.curvarDir:
                //System.out.println("ya");
                buffer.putInt(msg.getArg1());
                buffer.putInt(msg.getArg2());
                break;

            case iMensagem.curvarEsq:
                buffer.putInt(msg.getArg1());
                buffer.putInt(msg.getArg2());
                break;

            case iMensagem.vazia:
                buffer.putInt(0);
                System.out.println("limpei");
                //id = -1;
                break;
        }

        if (tipo != 4)
            putBuffer =++ putBuffer % nMensagens;
        msg.setId(id + 1);

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

}
```

Figura 1: Método enviarMensagem()

```
// recebe e converte numa Mensagem, lê e retorna mensagem
Mensagem receberMensagem() {

    int id, tipo, distancia, raio, angulo;
    boolean sincrono;

    buffer.asIntBuffer();
    //buffer.position(0);
    System.out.println("getbuffer id:" + getBuffer + ", nMensanges: " + nMensagens);
    buffer.position(getBuffer * 16);

    id   = buffer.getInt();
    tipo = buffer.getInt();

    Mensagem msg = null;

    switch (tipo) {
        case iMensagem.parar:
            sincrono = buffer.getInt() == 1;
            msg = new MensagemParar(id, tipo, sincrono);
            break;

        case iMensagem.reta:
            distancia = buffer.getInt();
            msg = new MensagemReta(id, tipo, distancia);
            break;

        case iMensagem.curvarDir:
            raio = buffer.getInt();
            angulo = buffer.getInt();
            msg = new MensagemCurvar(id, tipo, raio, angulo);
            break;

        case iMensagem.curvarEsq:
            raio = buffer.getInt();
            angulo = buffer.getInt();
            msg = new MensagemCurvar(id, tipo, raio, angulo);
            break;

        case iMensagem.vazia:
            msg = new Mensagem(id, tipo, 0, 0);
            break;
    }

    if(tipo==4)
        getBuffer =++ getBuffer % nMensagens;

    return msg;

}
```

Figura 2: Método receberMensagem()

```
public boolean abrirCanalEscritor(String Filename) {
    // cria um ficheiro
    ficheiro = new File(Filename);

    //cria um canal de comunicação de leitura e escrita
    try {
        canal = new RandomAccessFile(ficheiro, "rw").getChannel();
    } catch (FileNotFoundException e) {
        return false;
    }

    // mapeia para memória o conteúdo do ficheiro
    try {
        buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0, BUFFER_MAX);

        //Inicializar apontadores - put e get buffer a zero
        getBuffer = 0;
        putBuffer = 0;

        //Inicializar buffer - dizer que todos os blocos são vazios
        inicializarBuffer();

    } catch (IOException e) {
        return false;
    }

    return true;
}

public boolean abrirCanalLeitor(String Filename) {
    // cria um ficheiro
    ficheiro = new File(Filename);

    //cria um canal de comunicação de leitura e escrita
    try {
        canal = new RandomAccessFile(ficheiro, "rw").getChannel();
    } catch (FileNotFoundException e) {
        return false;
    }

    // mapeia para memória o conteúdo do ficheiro
    try {
        buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0, BUFFER_MAX);

        //Inicializar apontadores - put e get buffer a zero
        getBuffer = 0;
        putBuffer = 0;

    } catch (IOException e) {
        return false;
    }

    return true;
```

Figura 3: Métodos abrirCanalEscritor() e abrirCanalLeitor()

3 Conclusões

O canal de comunicação e a mensagem são um passo crucial no desenvolvimento deste trabalho. É também o passo subjetivamente mais difícil deste projeto que requer algum trabalho fora do período da aula. Vários problemas foram surgindo na altura em que foi feito o teste de funcionamento, maioritariamente com o estado vazio.

Dúvidas surgiram relacionadas com o propósito deste estado relacionadas com o facto de se este era lido pelo Súbdito ou até o facto deste alterar os valores dos **id** e **type** das mensagens seguintes. Na nossa implementação com a máquina de estados, foi importante definir que se o tipo for 4 (o tipo da mensagem fazia) que o buffer não deveria incrementar e que esta não deveria ser recebida pelo Súbdito.

4 Bibliografia

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

5 Código Java GUI_Base, GUI_Subdito, GUI_Rei

```
Classe App_Subdito

package ptrabalho;

public class App_Subdito
{
    @SuppressWarnings("unused")
    private GUI_Subdito gui;
    private BD_Subdito bd;

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

    public BD_Subdito getBD()
    {
        return bd;
    }

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

public class BD_Subdito
{
    private RobotLegoEV3 robot;
    private boolean terminar;
    private boolean ligado;
    private int distance;
    private int angulo;
    private float raio;
    private String nome;

    public BD_Subdito()
    {
        robot = new RobotLegoEV3();
        terminar = false;
        ligado = false;
        distance = 30;
```

```
59          angulo = 45;
60          raio = 5;
61      }
62
63      public RobotLegoEV3 getRobot()
64      {
65          return robot;
66      }
67
68      public boolean getTerminar()
69      {
70          return terminar;
71      }
72
73      public void setTerminar(boolean b)
74      {
75          terminar = b;
76      }
77
78      public boolean isLigado()
79      {
80          return ligado;
81      }
82
83      public void setLigado(boolean b)
84      {
85          ligado = b;
86      }
87
88      public int getDist()
89      {
90          return distance;
91      }
92
93      public void setDist(int i)
94      {
95          distance = i;
96      }
97
98      public int getAng()
99      {
100         return angulo;
101     }
102
103     public void setAng(int i)
104     {
105         angulo = i;
106     }
107
108     public float getRaio()
109     {
110         return raio;
111     }
112
113     public void setRaio(float i)
114     {
115         raio = i;
116     }
117
118     public void setNome(String n)
```

```
119     {
120         nome = n;
121     }
122 
123     public String getNome()
124     {
125         return nome;
126     }
127 
128 }
129 
130 GUI_Subdito
131 
132 package ptrabalho;
133 
134 import java.awt.EventQueue;
135 
136 import javax.swing.BorderFactory;
137 import javax.swing.JFrame;
138 import javax.swing.JPanel;
139 import javax.swing.border.Border;
140 import javax.swing.border.EmptyBorder;
141 import javax.swing.border.LineBorder;
142 import javax.swing.border.TitledBorder;
143 
144 import java.awt.Color;
145 import javax.swing.JLabel;
146 import java.awt.Font;
147 import java.awt.event.ActionEvent;
148 import java.awt.event.ActionListener;
149 
150 import javax.swing.SwingConstants;
151 import javax.swing.JTextField;
152 import javax.swing.JRadioButton;
153 
154 public class GUI_Subdito extends GUI_Base
155 {
156 
157   private JPanel contentPane;
158   private JTextField txtNome;
159   private BD_Subdito bd;
160 
161   /**
162    * Launch the application.
163    */
164   public GUI_Subdito()
165   {
166     super();
167     EventQueue.invokeLater(new Runnable()
168     {
169       public void run()
170       {
171         try
172         {
173           init_GUI_Subdito();
174 
175         } catch (Exception e)
176         {
177           e.printStackTrace();
178         }
```

```
179       }
180     });
181   }
182 
183   /**
184    * Create the frame.
185    */
186   public void init_GUI_Subdito()
187   {
188     setTitle("Trabalho 1 - GUI Subdito");
189     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
190     setBounds(100, 100, 754, 695);
191 
192 
193     JLabel lblNomeDoRobot = new JLabel("Nome do robot");
194     lblNomeDoRobot.setHorizontalAlignment(SwingConstants.LEFT);
195     lblNomeDoRobot.setFont(new Font("Arial", Font.BOLD, 12));
196     lblNomeDoRobot.setBounds(30, 61, 103, 25);
197     getContentPane().add(lblNomeDoRobot);
198 
199     btnFrt.addActionListener(new ActionListener()
200     {
201       public void actionPerformed(ActionEvent e)
202       {
203         bd.getRobot().Reta(bd.getDist());
204             bd.getRobot().Parar(false);
205       }
206     });
207 
208     btnEsq.addActionListener(new ActionListener()
209     {
210       public void actionPerformed(ActionEvent e)
211       {
212         bd.getRobot().CurvarEsquerda(bd.getRaio(),bd.getAng());
213             bd.getRobot().Parar(false);
214       }
215     });
216 
217     btnDir.addActionListener(new ActionListener()
218     {
219       public void actionPerformed(ActionEvent e)
220       {
221         bd.getRobot().CurvarDireita(bd.getRaio(),bd.getAng());
222             bd.getRobot().Parar(false);
223       }
224     });
225 
226     btnTras.addActionListener(new ActionListener()
227     {
228       public void actionPerformed(ActionEvent e)
229       {
230         bd.getRobot().Reta(-(bd.getDist()));
231             bd.getRobot().Parar(false);
232       }
233     });
234 
235     btnParar.addActionListener(new ActionListener()
236     {
237       public void actionPerformed(ActionEvent e)
238       {
```

```
239         bd.getRobot().Parar(true);
240       }
241     });
242 
243     JRadioButton rdbtnAbrirFechar = new JRadioButton("Abrir /
    Fechar Robot");
244     rdbtnAbrirFechar.addActionListener(new ActionListener()
245     {
246       public void actionPerformed(ActionEvent e) {
247         if (bd.isLigado())
248             {
249                 bd.getRobot().CloseEV3();
250                 bd.setLigado(false);
251             } else
252             {
253                 bd.setLigado(bd.getRobot().OpenEV3(bd.getNome()));
254             }
255       }
256     });
257 
258     txtDist.addActionListener(new ActionListener()
259     {
260       public void actionPerformed(ActionEvent e)
261       {
262         bd.setDist(Integer.parseInt(txtDist.getText()));
263       }
264     });
265 
266     txtAng.addActionListener(new ActionListener()
267     {
268       public void actionPerformed(ActionEvent e)
269       {
270         bd.setAng(Integer.parseInt(txtAng.getText()));
271       }
272     });
273 
274     txtRaio.addActionListener(new ActionListener()
275     {
276       public void actionPerformed(ActionEvent e)
277       {
278         bd.setRaio(Float.parseFloat(txtRaio.getText()));
279       }
280     });
281 
282     txtNome = new JTextField();
283     txtNome.addActionListener(new ActionListener()
284     {
285       public void actionPerformed(ActionEvent e)
286       {
287         bd.setNome(txtNome.getText());
288       }
289     });
290 
291 
292     txtNome.setFont(new Font("Arial", Font.BOLD, 12));
293     txtNome.setColumns(10);
294     txtNome.setBounds(143, 61, 279, 25);
295     getContentPane().add(txtNome);
296 
297 
```

```
298
299     rdbtnAbrirFechar.setFont(new Font("Arial", Font.BOLD, 12));
300     rdbtnAbrirFechar.setBounds(455, 61, 158, 25);
301     getContentPane().add(rdbtnAbrirFechar);
302
303
304     Border bords_robot= BorderFactory.createLineBorder(new Color
        (0,0,0),1);
305     TitledBorder borda_robot = BorderFactory.createTitledBorder(
        bords_robot, "Robot");
306     JPanel panel = new JPanel();
307     panel.setName("Canal de Comunicação");
308     panel.setBorder(new LineBorder(new Color(0, 0, 0)));
309     panel.setBounds(10, 34, 719, 66);
310     panel.setBorder(borda_robot);
311     getContentPane().add(panel);
312     contentPane = new JPanel();
313     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
314
315     setVisible(true);
316   }
317
318 }
319
320 Classe GUI_Rei
321
322 package ptrabalho;
323
324 import java.awt.Color;
325 import java.awt.EventQueue;
326 import java.awt.Font;
327
328 import javax.swing.BorderFactory;
329 import javax.swing.JButton;
330 import javax.swing.JFrame;
331 import javax.swing.JPanel;
332 import javax.swing.border.Border;
333 import javax.swing.border.EmptyBorder;
334 import javax.swing.border.LineBorder;
335 import javax.swing.border.TitledBorder;
336
337 public class GUI_Rei extends GUI_Base
338 {
339
340   private JPanel contentPane;
341
342   /**
343    * Launch the application.
344    */
345   public static void main(String[] args)
346   {
347     EventQueue.invokeLater(new Runnable()
348     {
349       public void run() {
350         try
351         {
352           GUI_Rei frame = new GUI_Rei();
353           frame.setVisible(true);
354         } catch (Exception e)
355         {
```

```
356             e.printStackTrace();
357           }
358         }
359       });
360   }
361
362   /**
363    * Create the frame.
364    */
365   public GUI_Rei()
366   {
367     super();
368     setTitle("Trabalho 1 - GUI Rei");
369     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
370     setBounds(100, 100, 754, 697);
371
372     Border cost = BorderFactory.createLineBorder(new Color(0,0,0)
      ,1);
373     TitledBorder borda_rei = BorderFactory.createTitledBorder(cost,
     "Controle do Robot em Modo Automático");
374     JPanel panel_1_1 = new JPanel();
375     panel_1_1.setLayout(null);
376     panel_1_1.setName("Controle do Robot em Modo Automático");
377     panel_1_1.setBorder(new LineBorder(new Color(0, 0, 0)));
378     panel_1_1.setBounds(10, 34, 719, 66);
379     panel_1_1.setBorder(borda_rei);
380     getContentPane().add(panel_1_1);
381
382     JButton btnNewButton_2 = new JButton("8 Comandos Aleatórios");
383     btnNewButton_2.setFont(new Font("Arial", Font.BOLD, 12));
384     btnNewButton_2.setBounds(10, 26, 300, 25);
385     panel_1_1.add(btnNewButton_2);
386
387     JButton btnNewButton_2_1 = new JButton("16 Comandos Aleatórios"
     );
388     btnNewButton_2_1.setFont(new Font("Arial", Font.BOLD, 12));
389     btnNewButton_2_1.setBounds(385, 26, 300, 25);
390     panel_1_1.add(btnNewButton_2_1);
391     contentPane = new JPanel();
392     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
393
394
395   }
396 }
397
398 Classe GUI_Base
399
400 package ptrabalho;
401
402 import java.awt.EventQueue;
403
404 import javax.swing.JFrame;
405 import javax.swing.JPanel;
406 import javax.swing.border.Border;
407 import javax.swing.border.EmptyBorder;
408 import javax.swing.BoxLayout;
409 import java.awt.GridLayout;
410
411 import javax.swing.BorderFactory;
412 import javax.swing.Box;
```

```
413 import java.awt.CardLayout;
414 import javax.swing.JTextField;
415 import java.awt.FlowLayout;
416 import javax.swing.JLabel;
417 import javax.swing.SwingConstants;
418 import java.awt.Font;
419 import javax.swing.JButton;
420 import javax.swing.JToggleButton;
421 import javax.swing.JSpinner;
422 import javax.swing.JRadioButton;
423 import javax.swing.JSeparator;
424 import javax.swing.JComboBox;
425 import javax.swing.JTextPane;
426 import java.awt.Panel;
427 import java.awt.Color;
428 import java.awt.Canvas;
429 import javax.swing.border.LineBorder;
430 import javax.swing.border.TitledBorder;
431 import javax.swing.JCheckBox;
432 import java.awt.event.ActionListener;
433 import java.awt.event.ActionEvent;
434 import javax.swing.JTextArea;
435
436 public class GUI_Base extends JFrame
437 {
438
439   private JPanel contentPane;
440   private JTextField textField;
441   protected JTextField txtRaio;
442   protected JTextField txtAng;
443   protected JTextField txtDist;
444   protected JButton btnFrt;
445   protected JButton btnEsq;
446   protected JButton btnDir;
447   protected JButton btnParar;
448   protected JButton btnTras;
449   protected JTextArea txtLog;
450
451   /**
452    * Launch the application.
453    */
454   public static void main(String[] args)
455   {
456     EventQueue.invokeLater(new Runnable()
457     {
458       public void run()
459       {
460         try
461         {
462           GUI_Base frame = new GUI_Base();
463           frame.setVisible(true);
464         } catch (Exception e)
465         {
466           e.printStackTrace();
467         }
468       }
469     });
470   }
471
472   /**
```

```
473     * Create the frame.
474     */
475    public GUI_Base()
476    {
477      setTitle("Trabalho 1 - GUI Base");
478      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
479      setBounds(100, 100, 754, 691);
480      contentPane = new JPanel();
481      contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
482
483      setContentPane(contentPane);
484      contentPane.setLayout(null);
485
486      textField = new JTextField();
487      textField.setFont(new Font("Arial", Font.BOLD, 12));
488      textField.setBounds(143, 156, 494, 25);
489      contentPane.add(textField);
490      textField.setColumns(10);
491
492      JLabel lblNewLabel = new JLabel("Ficheiro do Canal");
493      lblNewLabel.setFont(new Font("Arial", Font.BOLD, 12));
494      lblNewLabel.setHorizontalAlignment(SwingConstants.LEFT);
495      lblNewLabel.setBounds(30, 156, 103, 25);
496      contentPane.add(lblNewLabel);
497
498      JButton btnNewButton = new JButton("...");
499      btnNewButton.setFont(new Font("Arial", Font.BOLD, 12));
500      btnNewButton.setBounds(647, 157, 61, 25);
501      contentPane.add(btnNewButton);
502
503      JLabel lblNMsg = new JLabel("N   msg");
504      lblNMsg.setHorizontalAlignment(SwingConstants.LEFT);
505      lblNMsg.setFont(new Font("Arial", Font.BOLD, 12));
506      lblNMsg.setBounds(30, 200, 103, 25);
507      contentPane.add(lblNMsg);
508
509      JSpinner spinner = new JSpinner();
510      spinner.setFont(new Font("Arial", Font.BOLD, 12));
511      spinner.setBounds(143, 200, 50, 25);
512      contentPane.add(spinner);
513
514      JRadioButton rdbtnNewRadioButton = new JRadioButton("Abrir
       Canal");
515      rdbtnNewRadioButton.setFont(new Font("Arial", Font.BOLD, 12));
516      rdbtnNewRadioButton.setBounds(459, 200, 103, 25);
517      contentPane.add(rdbtnNewRadioButton);
518
519      Border simpb = BorderFactory.createLineBorder(new Color(0,0,0)
       ,1);
520      TitledBorder border_simp = BorderFactory.createTitledBorder(
       simpb,"Canal de Comunicação");
521      JPanel panel = new JPanel();
522      panel.setName("Canal de Comunicação");
523      panel.setBorder(new LineBorder(new Color(0, 0, 0)));
524      panel.setBounds(10, 128, 719, 113);
525      panel.setBorder(border_simp);
526      contentPane.add(panel);
527
528      JLabel lblRaio = new JLabel("Raio");
529      lblRaio.setHorizontalAlignment(SwingConstants.LEFT);
```

```
530     lblRaio.setFont(new Font("Arial", Font.BOLD, 12));
531     lblRaio.setBounds(30, 270, 103, 25);
532     contentPane.add(lblRaio);
533 
534     txtRaio = new JTextField();
535 
536     txtRaio.setFont(new Font("Arial", Font.BOLD, 12));
537     txtRaio.setColumns(10);
538     txtRaio.setBounds(143, 270, 100, 23);
539     contentPane.add(txtRaio);
540 
541     JLabel lblngulo = new JLabel("Ângulo");
542     lblngulo.setHorizontalAlignment(SwingConstants.LEFT);
543     lblngulo.setFont(new Font("Arial", Font.BOLD, 12));
544     lblngulo.setBounds(30, 300, 103, 25);
545     contentPane.add(lblngulo);
546 
547     JLabel lblDistncia = new JLabel("Distância");
548     lblDistncia.setHorizontalAlignment(SwingConstants.LEFT);
549     lblDistncia.setFont(new Font("Arial", Font.BOLD, 12));
550     lblDistncia.setBounds(30, 330, 103, 25);
551     contentPane.add(lblDistncia);
552 
553     txtAng = new JTextField();
554     txtAng.setFont(new Font("Arial", Font.BOLD, 12));
555     txtAng.setColumns(10);
556     txtAng.setBounds(143, 300, 100, 23);
557     contentPane.add(txtAng);
558 
559     txtDist = new JTextField();
560     txtDist.setFont(new Font("Arial", Font.BOLD, 12));
561     txtDist.setColumns(10);
562     txtDist.setBounds(143, 330, 100, 23);
563     contentPane.add(txtDist);
564 
565     btnParar = new JButton("Parar");
566     btnParar.setFont(new Font("Arial", Font.BOLD, 12));
567     btnParar.setBounds(470, 297, 100, 25);
568     contentPane.add(btnParar);
569 
570     btnFrt = new JButton("Frente");
571     btnFrt.setFont(new Font("Arial", Font.BOLD, 12));
572     btnFrt.setBounds(470, 269, 100, 25);
573     contentPane.add(btnFrt);
574 
575     btnDir = new JButton("Direita");
576     btnDir.setFont(new Font("Arial", Font.BOLD, 12));
577     btnDir.setBounds(575, 298, 100, 25);
578     contentPane.add(btnDir);
579 
580     btnEsq = new JButton("Esquerda");
581     btnEsq.setFont(new Font("Arial", Font.BOLD, 12));
582     btnEsq.setBounds(365, 297, 100, 25);
583     contentPane.add(btnEsq);
584 
585     btnTras = new JButton("Tras");
586     btnTras.setFont(new Font("Arial", Font.BOLD, 12));
587     btnTras.setBounds(470, 325, 100, 25);
588     contentPane.add(btnTras);
589 
```

```
590     Border borda_cont_robot = BorderFactory.createLineBorder(new
        Color(0,0,0),1);
591     TitledBorder borda1 = BorderFactory.createTitledBorder(
        borda_cont_robot,"Controle do Robot");
592     JPanel panel_1 = new JPanel();
593     panel_1.setFont(new Font("Arial", Font.BOLD, 12));
594     panel_1.setName("Controle do Robot");
595     panel_1.setBorder(new LineBorder(new Color(0, 0, 0)));
596     panel_1.setBounds(10, 251, 719, 113);
597     panel_1.setBorder(borda1);
598     contentPane.add(panel_1);
599
600     JCheckBox chckbxNewCheckBox = new JCheckBox("Ativar / Desativar
        Comportamento");
601     chckbxNewCheckBox.setFont(new Font("Arial", Font.BOLD, 12));
602     chckbxNewCheckBox.setBounds(10, 365, 262, 45);
603     contentPane.add(chckbxNewCheckBox);
604
605     JLabel lblLog = new JLabel("Log");
606     lblLog.setHorizontalAlignment(SwingConstants.LEFT);
607     lblLog.setFont(new Font("Arial", Font.BOLD, 12));
608     lblLog.setBounds(10, 429, 103, 25);
609     contentPane.add(lblLog);
610
611     JButton btnLimpaLog = new JButton("Limpar Log");
612     btnLimpaLog.setFont(new Font("Arial", Font.BOLD, 12));
613     btnLimpaLog.setBounds(10, 630, 719, 25);
614     contentPane.add(btnLimpaLog);
615
616     JTextArea txtLog = new JTextArea();
617     txtLog.setBounds(30, 459, 683, 148);
618     contentPane.add(txtLog);
619   }
620
621
622 }
623
624 package ptrabalho;
625
626 public class CanalComunicacaoConsistente extends CanalComunicacao{
627
628
629     public CanalComunicacaoConsistente(int n) {
630     super(n);
631     // TODO Auto-generated constructor stub
632   }
633
634   public boolean GetandSetWrite(Mensagem msg) {
635         Mensagem mensagem = GetandSetRead();
636         //System.out.println(mensagem);
637         try {
638           if(mensagem == null)
639             return false;
640             if(mensagem.tipo == iMensagem.vazia) {
641                 System.out.println("entrou, no buffer: " + mensagem
        + "enviei msg = " + msg);
642
643                 fl = canal.lock();
644
645                 enviarMensagem(msg);
```

```
646
647                 fl.release();
648                 return true;
649             }else System.out.println("NÃO entrou, no buffer: " +
    mensagem + "quero enviar msg = " + msg);
650         } catch (Exception e) {
651             e.printStackTrace();
652         }
653
654         return false;
655     }
656
657     public Mensagem GetandSetRead(){
658
659         try {
660             fl = canal.lock();
661
662             Mensagem mensagem = receberMensagem();
663
664             //limparLida();
665
666             fl.release();
667
668             return mensagem;
669
670         } catch (Exception e) {
671             e.printStackTrace();
672         }
673
674         return null;
675     }
676
677
678
679     /*public static void main (String[] args) {
680         CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente();
681
682         MensagemParar msgParar = new MensagemParar(0, 0, false);
    //int id, int tipo, boolean sincrono
683         MensagemReta msg     = new MensagemReta(1,1,20);          //
    int id, int tipo, int dist
684         MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15);
    //int id, int tipo, int raio, int ang
685         MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);
686
687         //Testar
688         cc.abrirCanal("comunicacao.dat");
689
690
691         cc.GetandSetWrite(msg);
692         System.out.println(cc.receberMensagem());
693         cc.GetandSetWrite(msgCD);
694         cc.GetandSetWrite(msgCD);
695         cc.GetandSetWrite(msgCE);
696         cc.GetandSetWrite(msgCD);
697         cc.GetandSetWrite(msgCD);
698         cc.GetandSetWrite(msgParar);
699         cc.GetandSetWrite(msgParar);
700         cc.GetandSetWrite(msgCD);
```

```
701         cc.GetandSetWrite(msgCD);
702         cc.GetandSetWrite(msgCD);
703         cc.GetandSetWrite(msgCD);
704         cc.GetandSetWrite(msgParar);
705         cc.GetandSetWrite(msgParar);
706         cc.GetandSetWrite(msgCD);
707         cc.GetandSetWrite(msgCD);
708         cc.GetandSetWrite(msgCD);
709         cc.GetandSetWrite(msgParar);
710         cc.GetandSetWrite(msgParar);
711 
712         cc.fecharCanal();
713     }*/
714 
715 }
716 package ptrabalho;
717 import java.io.*;
718 import java.nio.MappedByteBuffer;
719 import java.nio.channels.FileChannel;
720 import java.nio.channels.FileLock;
721 
722 public class CanalComunicacao {
723     FileLock fl;
724 
725     // ficheiro
726     private File ficheiro;
727 
728     // canal que liga o conteúdo do ficheiro ao Buffer
729     protected FileChannel canal;
730 
731     // buffer
732     private MappedByteBuffer buffer;
733 
734     // dimensão máxima em bytes do buffer = (N   grup + 1) * 4
    bytes (int) * 4 ints
735     //final int nGrupo = 15;
736     //final int BUFFER_MAX = (nGrupo + 1) * 4 * 4;
737     protected int nMensagens = 8;
738     protected int BUFFER_MAX = nMensagens*16;
739     protected int putBuffer, getBuffer;
740 
741 
742     //Construtor onde se cria o canal
743     public CanalComunicacao(int n) {
744         ficheiro = null;
745         canal    = null;
746         buffer   = null;
747         nMensagens = n;
748         BUFFER_MAX = nMensagens*16;
749     }
750 
751     public boolean abrirCanalEscritor(String Filename) {
752         // cria um ficheiro
753         ficheiro = new File(Filename);
754 
755         //cria um canal de comunicação de leitura e escrita
756         try {
757             canal = new RandomAccessFile(ficheiro, "rw").getChannel
    ();
758         } catch (FileNotFoundException e) {
```

```
759             return false;
760         }
761 
762         // mapeia para memória o conteúdo do ficheiro
763         try {
764             buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
    BUFFER_MAX);
765 
766             //Inicializar apontadores - put e get buffer a zero
767             getBuffer = 0;
768             putBuffer = 0;
769 
770             //Inicializar buffer - dizer que todos os blocos são
    vazios
771             inicializarBuffer();
772 
773         } catch (IOException e) {
774             return false;
775         }
776 
777         return true;
778     }
779 
780     public boolean abrirCanalLeitor(String Filename) {
781         // cria um ficheiro
782         ficheiro = new File(Filename);
783 
784         //cria um canal de comunicação de leitura e escrita
785         try {
786             canal = new RandomAccessFile(ficheiro, "rw").getChannel
    ();
787         } catch (FileNotFoundException e) {
788             return false;
789         }
790 
791         // mapeia para memória o conteúdo do ficheiro
792         try {
793             buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
    BUFFER_MAX);
794 
795             //Inicializar apontadores - put e get buffer a zero
796             getBuffer = 0;
797             putBuffer = 0;
798 
799         } catch (IOException e) {
800             return false;
801         }
802 
803         return true;
804     }
805 
806     // recebe e converte numa Mensagem, lê e retorna mensagem
807     Mensagem receberMensagem() {
808 
809         int id, tipo, distancia, raio, angulo;
810         boolean sincrono;
811 
812         buffer.asIntBuffer();
813         //buffer.position(0);
814         System.out.println("getbuffer id:" + getBuffer + ",
```

```
   nMensanges: " + nMensagens);
815        buffer.position(getBuffer * 16);
816
817        id   = buffer.getInt();
818        tipo = buffer.getInt();
819
820        Mensagem msg = null;
821
822        switch (tipo) {
823            case iMensagem.parar:
824                sincrono = buffer.getInt() == 1;
825                msg = new MensagemParar(id, tipo, sincrono);
826                break;
827
828            case iMensagem.reta:
829                distancia = buffer.getInt();
830                msg = new MensagemReta(id, tipo, distancia);
831                break;
832
833            case iMensagem.curvarDir:
834                raio = buffer.getInt();
835                angulo = buffer.getInt();
836                msg = new MensagemCurvar(id, tipo, raio, angulo);
837                break;
838
839            case iMensagem.curvarEsq:
840                raio = buffer.getInt();
841                angulo = buffer.getInt();
842                msg = new MensagemCurvar(id, tipo, raio, angulo);
843                break;
844
845            case iMensagem.vazia:
846                msg = new Mensagem(id, tipo, 0, 0);
847                break;
848        }
849
850        if(tipo==4)
851          getBuffer =++ getBuffer % nMensagens;
852
853        return msg;
854
855    }
856
857 // recebe e converte numa Mensagem, lê e retorna mensagem
858    Mensagem receberMensagemLeitor() {
859
860        int id, tipo, distancia, raio, angulo;
861        boolean sincrono;
862
863        buffer.asIntBuffer();
864        //buffer.position(0);
865        System.out.println("getbuffer: " + getBuffer + ",
   nMensanges: " + nMensagens);
866        buffer.position(getBuffer * 16);
867
868        id   = buffer.getInt();
869        tipo = buffer.getInt();
870
871        Mensagem msg = null;
872
```

```
873         switch (tipo) {
874             case iMensagem.parar:
875                 sincrono = buffer.getInt() == 1;
876                 msg = new MensagemParar(id, tipo, sincrono);
877                 break;
878 
879             case iMensagem.reta:
880                 distancia = buffer.getInt();
881                 msg = new MensagemReta(id, tipo, distancia);
882                 break;
883 
884             case iMensagem.curvarDir:
885                 raio = buffer.getInt();
886                 angulo = buffer.getInt();
887                 msg = new MensagemCurvar(id, tipo, raio, angulo);
888                 break;
889 
890             case iMensagem.curvarEsq:
891                 raio = buffer.getInt();
892                 angulo = buffer.getInt();
893                 msg = new MensagemCurvar(id, tipo, raio, angulo);
894                 break;
895 
896             case iMensagem.vazia:
897                 msg = new Mensagem(id, tipo, 0, 0);
898                 break;
899         }
900 
901         if(tipo!=4)
902           getBuffer =++ getBuffer % nMensagens;
903 
904         return msg;
905 
906     }
907 
908     // envia uma Mensagem como um conjunto de ints
909 // envia uma Mensagem como um conjunto de ints
910     void enviarMensagem(Mensagem msg) {
911 
912         try {
913             //buffer.position(0);
914             buffer.position(putBuffer * 16);
915 
916             // Obter ID e escrevê-lo no buffer
917             int id = msg.getId();
918             buffer.putInt(id);
919 
920             // Obter Tipo e escrevê-lo no buffer
921             int tipo = msg.getTipo();
922             buffer.putInt(tipo);
923 
924             // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
925             switch (tipo) {
926                 case iMensagem.parar:
927                     buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
928                     break;
929 
930                 case iMensagem.reta:
```

```
931                     buffer.putInt(msg.getArg1());
932                     break;
933 
934                 case iMensagem.curvarDir:
935                   //System.out.println("yo");
936                     buffer.putInt(msg.getArg1());
937                     buffer.putInt(msg.getArg2());
938                     break;
939 
940                 case iMensagem.curvarEsq:
941                     buffer.putInt(msg.getArg1());
942                     buffer.putInt(msg.getArg2());
943                     break;
944 
945                 case iMensagem.vazia:
946                     buffer.putInt(0);
947                     System.out.println("limpei");
948                     //id = -1;
949                     break;
950             }
951 
952             if (tipo != 4)
953               putBuffer =++ putBuffer % nMensagens;
954             msg.setId(id + 1);
955 
956         }catch(Exception e) {
957             e.printStackTrace();
958         }
959 
960     }
961 
962     void enviarMensagemClean(Mensagem msg) {
963 
964         try {
965             //buffer.position(0);
966             buffer.position(putBuffer * 16);
967 
968             // Obter ID e escrevê-lo no buffer
969             int id = msg.getId();
970             buffer.putInt(id);
971 
972             // Obter Tipo e escrevê-lo no buffer
973             int tipo = msg.getTipo();
974             buffer.putInt(tipo);
975 
976             // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
977             switch (tipo) {
978                 case iMensagem.parar:
979                     buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
980                     break;
981 
982                 case iMensagem.reta:
983                     buffer.putInt(msg.getArg1());
984                     break;
985 
986                 case iMensagem.curvarDir:
987                   System.out.println("yo");
988                     buffer.putInt(msg.getArg1());
```

```
989                     buffer.putInt(msg.getArg2());
990                     break;
991
992                 case iMensagem.curvarEsq:
993                     buffer.putInt(msg.getArg1());
994                     buffer.putInt(msg.getArg2());
995                     break;
996
997                 case iMensagem.vazia:
998                     buffer.putInt(0);
999                     System.out.println("limpei");
1000                    //id = -1;
1001                    break;
1002            }
1003
1004            putBuffer =++ putBuffer % nMensagens;
1005            msg.setId(id + 1);
1006
1007        }catch(Exception e) {
1008            e.printStackTrace();
1009        }
1010
1011    }
1012
1013    // fecha o canal entre o buffer e o ficheiro
1014    void fecharCanal() {
1015        if (canal != null)
1016            try {
1017                canal.close();
1018            } catch (IOException e) {
1019                canal = null;
1020            }
1021    }
1022
1023    /**Colocar todos os slots com mensagens vazias*/
1024    //Mts duvidas, gestao do ID, so tipo e id
1025    void inicializarBuffer() {
1026
1027        //1 msg - 4 * 4 = 16 bytes
1028        MensagemVazia msg = new MensagemVazia(0, iMensagem.vazia);
1029        System.out.print(BUFFER_MAX);
1030
1031        for(int i = 0; i < nMensagens; i++) {
1032            enviarMensagemClean(msg);
1033        }
1034        buffer.clear();
1035        putBuffer=0;
1036        getBuffer=0;
1037
1038    }
1039
1040    void limparLida(){
1041        buffer.position(getBuffer * 16);
1042        buffer.putInt(0);
1043        buffer.putInt(iMensagem.vazia);
1044
1045    }
1046
1047
1048    public static void main(String[] args) {
```

```
1049
1050
1051        MensagemParar msgParar = new MensagemParar(0, 0, false);   //
    int id, int tipo, boolean sincrono
1052        MensagemReta msg     = new MensagemReta(1,1,20);          //
    int id, int tipo, int dist
1053        MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15);
    //int id, int tipo, int raio, int ang
1054        MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);
1055        MensagemVazia msgV = new MensagemVazia(4,4);
1056
1057
1058        CanalComunicacao cc = new CanalComunicacao(8);
1059        cc.abrirCanalEscritor("comunicacao.dat");
1060
1061        cc.enviarMensagem(msg);
1062        Mensagem msg1 =cc.receberMensagemLeitor();
1063        System.out.println(msg1);
1064        cc.enviarMensagem(msgV);
1065        System.out.println(cc.receberMensagemLeitor());
1066        cc.enviarMensagem(msgCD);
1067        System.out.println(cc.receberMensagemLeitor());
1068        cc.enviarMensagem(msgCE);
1069        System.out.println(cc.receberMensagemLeitor());
1070
1071        cc.fecharCanal();
1072        }
1073 }
1074
1075 package ptrabalho;
1076
1077 public interface iMensagem {
1078
1079     int parar          = 0;
1080     int reta           = 1;
1081     int curvarDir        = 2;
1082     int curvarEsq        = 3;
1083     int vazia             = 4;
1084     int pedir            = 5;
1085     int sensor            = 6;
1086     int workflow        = 7;
1087     int endWorkflow    = 8;
1088 }
1089
1090 package ptrabalho;
1091
1092 public class Mensagem implements iMensagem{
1093     int tipo, id, arg1, arg2;
1094
1095     public Mensagem(int id, int tipo, int arg1, int arg2) {
1096         this.tipo = tipo;
1097         this.id = id;
1098         this.arg1 = arg1;
1099         this.arg2 = arg2;
1100     }
1101
1102     public Mensagem() {
1103
1104     }
1105
```

```
1106
1107        @Override
1108        public String toString() {
1109            return "Mensagem{" +
1110                    " id= " + id +
1111                    " tipo=" + tipo +
1112                    " arg1=" + arg1 +
1113                    " arg2=" + arg2 +
1114                    '}';
1115        }
1116
1117        public int getTipo() {
1118            return tipo;
1119        }
1120
1121        public void setTipo(int tipo) {
1122            this.tipo = tipo;
1123        }
1124
1125        public int getId() {
1126            return id;
1127        }
1128
1129        public void setId(int id) {
1130            this.id = id;
1131        }
1132
1133        public int getArg1() {
1134            return arg1;
1135        }
1136
1137        public void setArg1(int arg) {
1138            this.arg1 = arg;
1139        }
1140
1141        public int getArg2() {
1142            return arg2;
1143        }
1144
1145        public void setArg2(int arg) {
1146            this.arg2 = id;
1147        }
1148
1149        public boolean equals(Mensagem mensagem) {
1150
1151            if(mensagem==null) return false;
1152
1153            return (this.getId() == mensagem.getId());
1154        }
1155    }
1156
1157    package ptrabalho;
1158
1159    public class MensagemCurvar extends Mensagem{
1160      int raio, ang;
1161
1162      public MensagemCurvar(int id, int tipo, int raio, int ang) {
1163        super(id, tipo, raio, ang);
1164        this.raio = raio;
1165        this.ang  = ang;
```

```
1166    }
1167
1168    public int getRaio() {
1169      return raio;
1170    }
1171
1172    public void setRaio(int raio) {
1173      this.raio = raio;
1174    }
1175
1176    public int getAng() {
1177      return ang;
1178    }
1179
1180    public void setAng(int ang) {
1181      this.ang = ang;
1182    }
1183  }
1184
1185  package ptrabalho;
1186
1187  public class MensagemParar extends Mensagem{
1188
1189    boolean sincrono;
1190
1191    public MensagemParar(int id, int tipo, boolean sincrono) {
1192      super(id, tipo, 0, 0);
1193      this.sincrono = sincrono;
1194    }
1195
1196    @Override
1197    public String toString() {
1198      return super.toString() +" " +
1199          "sincrono=" + sincrono +
1200          '}';
1201    }
1202
1203    public boolean isSincrono() {
1204      return sincrono;
1205    }
1206
1207    public void setSincrono(boolean sincrono) {
1208      this.sincrono = sincrono;
1209    }
1210
1211  }
1212
1213  package ptrabalho;
1214
1215  public class MensagemReta extends Mensagem{
1216    int dist;
1217
1218    public MensagemReta(int id, int tipo, int dist) {
1219      super(id, tipo, dist, 0);
1220      this.dist = dist;
1221    }
1222
1223    public MensagemReta(int id, int tipo, int dist, int arg2) {
1224      super(id, tipo, dist, 0);
1225      this.dist = dist;
```

```
1226    }
1227
1228    public int getDist() {
1229      return dist;
1230    }
1231
1232    public void setDist(int dist) {
1233      this.dist = dist;
1234    }
1235
1236    @Override
1237    public String toString() {
1238      return super.toString() + " distancia= " + dist + "}";
1239    }
1240  }
1241
1242  package ptrabalho;
1243
1244  public class MensagemVazia extends Mensagem{
1245      public MensagemVazia(int id, int tipo) {
1246          super(id, tipo, 0, 0);
1247      }
1248  }
```