Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Fundamentos de Sistemas Operativos - 2324SI

1º Trabalho Prático - Aula prática 6

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 Súbdito. O Súbdito, como no jogo "O Rei Manda"vai receber instruções, enviadas pelo Rei, através do canal de comunicação previamente desenvolvido, e vai realizá-las. O processo Súbdito está dividido em 3 classes: a BD_Subdito, a GUI_Subdito e a App_Subdito. As bases destas 3 classes já tinham sido criadas mas nesta aula vamos concretizá-las. A BD_Subdito vai extender da BD_Base, ou seja, permite-nos ter uma base de dados apenas para o processo Súbdito. O mesmo se passa para a GUI_Subdito que vai extender da GUI_Base. A App_Subdito vai conter uma máquina de estados que vai controlar o recebimento de mensagens através do canal de comunicação e a consequente execução das mesmas.

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 Súbdito.

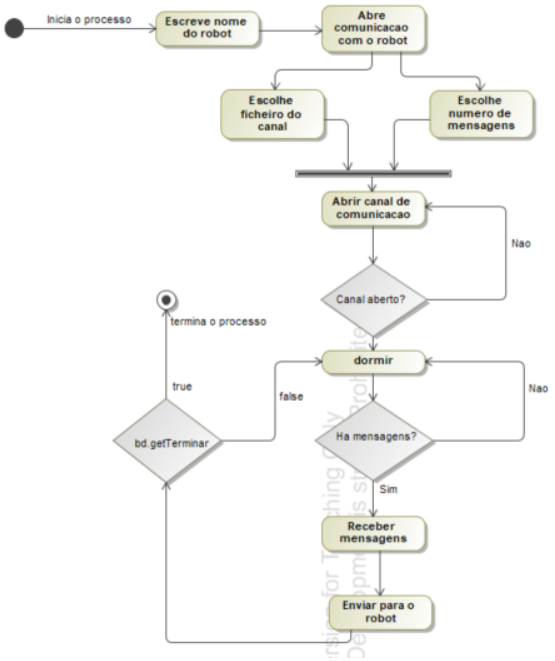

Figura 1: Diagrama de estados

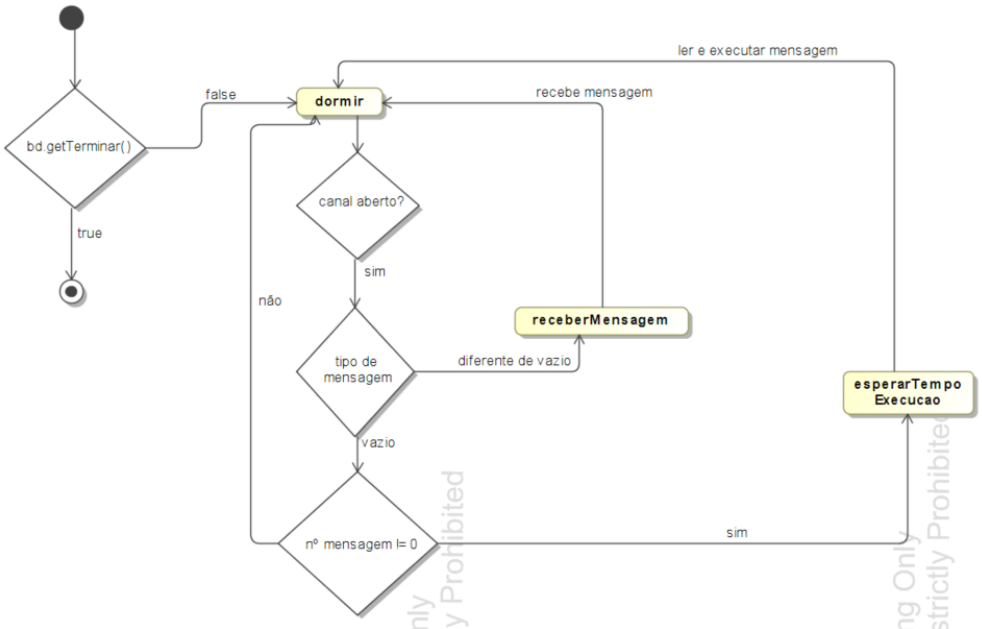

Figura 2: Diagrama de atividades

Nesta máquina de estados podemos observar que o processo do Súbdito dá inicio no momento em que o programa é corrido. Começamos por escrever o nome do robot e iniciamos a comunicação com o robot carregando no botão de ativar/desativar comportamento. Escolhemos o número de mensagens que podem ser enviadas pelo canal de comunicação e o ficheiro do cana que será igualmente escolhido no processo do Rei.

Abrimos o canal de comunicação e entramos num "sleep state"até que haja alguma mensagem para ser enviada para o robot. Se houver uma mensagem então ele recebe-a e envia-a para para o robot.

No segundo diagrama temos um diagrama de estados para o canal de comunicação no lado do Súbdito. Se o canal estiver aberto e a mensagem não for vazia, este recebe a mensagem e continua a receber até que a mensagem seja vazia. Se o número de mensagens guardadas for diferente de 0, ele espera o tempo de execução de cada uma delas, lê e vai enviando as mensagens para o Súbdito até que já não hajam mais mensagens para serem enviadas.

2.2 Processo Súbdito

2.2.1 BD_Subdito

Depois de termos criado o diagrama de atividades vamos atualizar as 3 classes do processo Súbdito. Na classe BD_Base criámos uma lista de mensagens assim como um método para adicionar uma mensagem e um método para devolver todas as mensagens. Estes métodos foram retirados da BD_Rei já que iam ser utilizados por ambos os processos. Esta lista assim como os seus métodos vão permitir ao Súbdito armazenar as instruções recebidas. Na BD_Subdito mantivemos as informações únicas ao processo, ou seja tudo relacionado com o robot como o seu nome, a sua instância e o estado da sua ligação.

2.2.2 GUI_Subdito

Na classe GUI_Subdito manteve-se tudo igual à aula 2, recapitulando, as diferenças entre esta e a GUI_Rei é o funcionamento dos botões e o facto desta estabelecer uma ligação direta com o robot. Os botões em vez de criarem instruções apenas cumprem-nos. Também não tem os botões dos comandos aleatórios.

2.2.3 App_Subdito

Finalmente vamos desenvolver a máquina de estados concebida no diagrama de atividades. Como descrito anteriormente esta vai nos permitir fazer a receção de mensagens através do canal de comunicação e a consequente realização da instrução pelo robot.

```
while(!bd.getTerminar()) {

    switch (state) {

    case receberMensagem:
        System.out.println("recebe");
        gui.txtLog.append(" Recebi = " + msg + "\n");
        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 {
                if(bd.getMensagens().size()!=0 && bd.isLigado()) {
                    state = esperarTempoExecucao;
                    break;
                }
                System.out.println("sleep");
                Thread.sleep(1000);
                break;}
            }else {
                Thread.sleep(1000);
                System.out.println("sleep");
                break;}
```

Figura 3: Máquina de Estados (1/2)

```
    case esperarTempoExecucao:
        msg = bd.getMensagens().get(0);
        System.out.println("esperaExec id:" + msg.getId());
        int tipo = msg.getTipo();
        System.out.println("Tipo:" + tipo);
        switch (tipo) {
            case RETA:
                gui.txtLog.append(" O robot avançou " + msg.getArg1() + "\n");
                bd.getRobot().Reta(msg.getArg1());
                bd.getRobot().Parar(false);
                Thread.sleep((long) ((Math.abs(msg.getArg1())) / 0.03));
                break;
            case CURVARDIR:
                gui.txtLog.append(" O robot virou direita com raio = " + msg.getArg1() +
                bd.getRobot().CurvarDireita(msg.getArg1(),msg.getArg2());
                bd.getRobot().Parar(false);
                Thread.sleep((long) ((msg.getArg1() * (msg.getArg2() * 0.017)) / 0.03));
                break;
            case CURVARESQ:
                gui.txtLog.append(" O robot virou esquerda com raio = " + msg.getArg1() +
                bd.getRobot().CurvarEsquerda(msg.getArg1(),msg.getArg2());
                bd.getRobot().Parar(false);
                Thread.sleep((long) ((msg.getArg1() * (msg.getArg2() * 0.017)) / 0.03));
                break;
        }
        bd.removeMensagem();
        state = dormir;
        break;
    }
}
System.out.println("sai");
Thread.sleep(100);
```

Figura 4: Máquina de Estados (2/2)

A máquina começa no estado dormir, onde se existir um canal de comunicação, procura e guarda uma mensagem se esta não for do tipo vazia passamos ao estado receberMensagem. Neste estado guardamos a mensagem na lista, notificamos o utilizador na consola e voltamos ao estado dormir. Se, no estado dormir, a mensagem for do tipo vazio, existirem mensagens na lista e o robot esteja ligado ao computador passamos ao estado esperarTempoExecucao.

Neste estado guardamos a primeira mensagem da lista e descobrimos qual o seu tipo: Reta, Curva Direita ou Curva Esquerda. De acordo com o seu tipo realizamos a instrução, notificamos o utilizador na consola e realizamos uma espera antes de mudar de estado. O tempo desta espera vai depender dos dados da instrução, ou seja, uma ação que demore mais tempo a realizar vai ter um maior tempo de espera até o robot poder realizar outra. Este mecanismo foi desenvolvido para evitarmos a destruição de mensagens pelo robot. Por fim removemos a mensagem da lista e retornamos ao estado dormir.

3 Conclusões

Esta fase do trabalho foi realizada com relativa facilidade pois as bases do processo já tinham sido realizadas na fase anterior, quer na máquina de estados quer na receção de mensagens já que tínhamos que testar o Rei. Para o trabalho ficar completo fica-nos a faltar a utilização dos jar assim como a possibilidade de enviar instruções aleatórias para andar para trás e para parar. Estes pormenores vão ser adicionados até à última aula.

4 Bibliografia

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

5 Código Java

```
Classe App_Subdito

package ptrabalho;
import java.lang.Math;

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;
    private final int RETA = 1;
    private final int CURVARDIR = 2;
    private final int CURVARESQ = 3;

    public App_Subdito(String[] args)
    {
      bd = new BD_Subdito();

      if (args.length == 2)
        {
        int nMensagens = Integer.parseInt(args[0]);
          if (nMensagens >= 8 && nMensagens <= 12)
            bd.setNMensagens(nMensagens);
          bd.setFile(args[1]);
        }
        gui = new GUI_Subdito(bd);
    }

    public BD_Subdito getBD()
    {
        return bd;
    }

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

              switch (state) {

              case receberMensagem:
                System.out.println("recebe");
                gui.txtLog.append(" Recebi = " + msg + "\n");
                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");
```

```
60
61                state = receberMensagem;
62                break;
63              }
64              else {
65                if(bd.getMensagens().size()!=0 && bd.isLigado()) {
66                      state = esperarTempoExecucao;
67                      break;
68                    }
69                System.out.println("sleep");
70                Thread.sleep(1000);
71                break;}
72              }else {
73                Thread.sleep(1000);
74                System.out.println("sleep");
75                break;}
76
77          case esperarTempoExecucao:
78            msg = bd.getMensagens().get(0);
79            System.out.println("esperaExec id:" + msg.getId());
80            int tipo = msg.getTipo();
81            System.out.println("Tipo:" + tipo);
82            switch (tipo) {
83              case RETA:
84                gui.txtLog.append(" O robot avançou " + msg.getArg1
    () + "\n");
85                bd.getRobot().Reta(msg.getArg1());
86                bd.getRobot().Parar(false);
87                Thread.sleep((long) ((Math.abs(msg.getArg1())) /
    0.03));
88                break;
89              case CURVARDIR:
90                gui.txtLog.append(" O robot virou direita com raio
    = " + msg.getArg1() +" e angulo = " + msg.getArg2() + "\n");
91                bd.getRobot().CurvarDireita(msg.getArg1(),msg.
    getArg2());
92                bd.getRobot().Parar(false);
93                Thread.sleep((long) ((msg.getArg1() * (msg.getArg2
    () * 0.017)) / 0.03));
94                break;
95              case CURVARESQ:
96                gui.txtLog.append(" O robot virou esquerda com raio
     = " + msg.getArg1() +" e angulo = " + msg.getArg2() + "\n");
97                  bd.getRobot().CurvarEsquerda(msg.getArg1(),msg.
    getArg2());
98                  bd.getRobot().Parar(false);
99                Thread.sleep((long) ((msg.getArg1() * (msg.getArg2
    () * 0.017)) / 0.03));
100               break;
101           }
102           bd.removeMensagem();
103           state = dormir;
104           break;
105         }
106       }
107       System.out.println("sai");
108           Thread.sleep(100);
109    }
110
```

```
111     public static void main(String[] args) throws
        InterruptedException
112     {
113         App_Subdito app = new App_Subdito(args);
114         System.out.println("A aplicação começou.");
115         app.run();
116         System.out.println("A aplicação terminou.");
117     }
118
119 }
120
121 Classe BD_Subdito
122
123 package ptrabalho;
124 import robot.RobotLegoEV3;
125
126 package ptrabalho;
127 //import robot.RobotLegoEV3;
128
129 package ptrabalho;
130 import robot.RobotLegoEV3;
131
132 public class BD_Subdito extends BD_Base
133 {
134     private RobotLegoEV3 robot;
135     private boolean terminar;
136     private boolean ligado;
137
138     private String nome;
139
140     public BD_Subdito()
141     {
142       super();
143         robot = new RobotLegoEV3();
144         terminar = false;
145         ligado = false;
146
147     }
148
149     public RobotLegoEV3 getRobot()
150     {
151         return robot;
152     }
153
154     public boolean getTerminar()
155     {
156         return terminar;
157     }
158
159     public void setTerminar(boolean b)
160     {
161         terminar = b;
162     }
163
164     public boolean isLigado()
165     {
166         return ligado;
167     }
168
169     public void setLigado(boolean b)
```

```
170     {
171         ligado = b;
172     }
173 
174     public void setNome(String n)
175     {
176         nome = n;
177     }
178 
179     public String getNome()
180     {
181         return nome;
182     }
183 
184 }
185 
186 package ptrabalho;
187 
188 import java.awt.EventQueue;
189 
190 import javax.swing.BorderFactory;
191 import javax.swing.JFileChooser;
192 import javax.swing.JFrame;
193 import javax.swing.JPanel;
194 import javax.swing.border.Border;
195 import javax.swing.border.EmptyBorder;
196 import javax.swing.border.LineBorder;
197 import javax.swing.border.TitledBorder;
198 
199 import java.awt.Color;
200 import javax.swing.JLabel;
201 import java.awt.Font;
202 import java.awt.event.ActionEvent;
203 import java.awt.event.ActionListener;
204 import java.awt.event.WindowAdapter;
205 import java.awt.event.WindowEvent;
206 
207 import javax.swing.SwingConstants;
208 import javax.swing.JTextField;
209 import javax.swing.JRadioButton;
210 import javax.swing.JTextArea;
211 
212 public class GUI_Subdito extends GUI_Base
213 {
214 
215   private JPanel contentPane;
216   private JTextField txtNome;
217   private App_Subdito app;
218   private BD_Subdito bd;
219 
220   /**
221    * Launch the application.
222    */
223   public GUI_Subdito(BD_Subdito bd)
224   {
225     super(bd);
226     EventQueue.invokeLater(new Runnable()
227     {
228       public void run()
229       {
```

```
230         try
231         {
232           init_GUI_Subdito(bd);
233 
234         } catch (Exception e)
235         {
236           e.printStackTrace();
237         }
238       }
239     });
240   }
241 
242   /**
243    * Create the frame.
244    */
245   public void init_GUI_Subdito(BD_Subdito bd)
246   {
247     setTitle("Trabalho 1 - GUI Subdito");
248     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
249     setBounds(100, 100, 754, 695);
250 
251 
252     JLabel lblNomeDoRobot = new JLabel("Nome do robot");
253     lblNomeDoRobot.setHorizontalAlignment(SwingConstants.LEFT);
254     lblNomeDoRobot.setFont(new Font("Arial", Font.BOLD, 12));
255     lblNomeDoRobot.setBounds(30, 61, 103, 25);
256     getContentPane().add(lblNomeDoRobot);
257 
258 
259     btnFrt.addActionListener(new ActionListener()
260     {
261       public void actionPerformed(ActionEvent e)
262       {
263         bd.getRobot().Reta(bd.getDist());
264             bd.getRobot().Parar(false);
265             txtLog.append(" O robot avançou " + bd.getDist() + "\n"
    );
266       }
267     });
268 
269     btnEsq.addActionListener(new ActionListener()
270     {
271       public void actionPerformed(ActionEvent e)
272       {
273         bd.getRobot().CurvarEsquerda(bd.getRaio(),bd.getAng());
274             bd.getRobot().Parar(false);
275             txtLog.append(" O robot virou esquerda com raio = " +
    bd.getRaio() +" e angulo = " + bd.getAng() + "\n");
276       }
277     });
278 
279     btnDir.addActionListener(new ActionListener()
280     {
281       public void actionPerformed(ActionEvent e)
282       {
283         bd.getRobot().CurvarDireita(bd.getRaio(),bd.getAng());
284             bd.getRobot().Parar(false);
285             txtLog.append(" O robot virou direita com raio = " + bd
    .getRaio() +" e angulo = " + bd.getAng() + "\n");
286       }
```

```
287     });
288
289     btnTras.addActionListener(new ActionListener()
290     {
291       public void actionPerformed(ActionEvent e)
292       {
293         bd.getRobot().Reta(-(bd.getDist()));
294             bd.getRobot().Parar(false);
295             txtLog.append(" O robot recuou " + bd.getDist() + "\n")
    ;
296       }
297     });
298
299     btnParar.addActionListener(new ActionListener()
300     {
301       public void actionPerformed(ActionEvent e)
302       {
303         bd.getRobot().Parar(true);
304         System.out.print(bd.getCanal().GetandSetReadLeitor());
305         txtLog.append(" O robot parou \n");
306         //txtLog.append("no limpa " + bd.getCanal().GetandSetRead()
    .toString() +"\n");
307
308       }
309     });
310
311
312     JRadioButton rdbtnAbrirFecharBlt = new JRadioButton("Abrir /
    Fechar Bluetooth");
313     rdbtnAbrirFecharBlt.addActionListener(new ActionListener()
314     {
315       public void actionPerformed(ActionEvent e) {
316         if (bd.isLigado())
317             {
318           System.out.println("Desligando...");
319                 bd.getRobot().CloseEV3();
320                 bd.setLigado(false);
321
322             } else
323             {
324               System.out.println("Ligando...");
325                 bd.setLigado(bd.getRobot().OpenEV3(bd.getNome()));
326                 System.out.println(bd.isLigado());
327                 if (!bd.isLigado())
328                   rdbtnAbrirFecharBlt.setSelected(false);
329
330                 //bd.setLigado(true);
331             }
332       }
333     });
334
335     rdbtnAbrirCanal.addActionListener(new ActionListener()
336     {
337       public void actionPerformed(ActionEvent e)
338       {
339         if (rdbtnAbrirCanal.isSelected())
340         {
341           CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente(bd.getNMensagens());
342           cc.abrirCanalLeitor(bd.getFile());
```

```
343           bd.setCanal(cc);
344           txtLog.append(" Canal de Comunicação aberto!! \n");
345 
346           //while(rdbtnAbrirCanal.isSelected())
347 
348             }
349         else {
350           bd.getCanal().fecharCanal();
351           bd.setCanal(null);
352           txtLog.append(" Canal de Comunicação fechado \n");
353         }
354       }
355     });
356 
357 
358     txtNome = new JTextField();
359     txtNome.addActionListener(new ActionListener()
360     {
361       public void actionPerformed(ActionEvent e)
362       {
363         bd.setNome(txtNome.getText());
364         txtLog.append(" O nome do Robot é " + bd.getNome() + "\n");
365       }
366     });
367 
368 
369     txtNome.setFont(new Font("Arial", Font.BOLD, 12));
370     txtNome.setColumns(10);
371     txtNome.setBounds(143, 61, 279, 25);
372     getContentPane().add(txtNome);
373 
374 
375 
376     rdbtnAbrirFecharBlt.setFont(new Font("Arial", Font.BOLD, 12));
377     rdbtnAbrirFecharBlt.setBounds(455, 61, 158, 25);
378     getContentPane().add(rdbtnAbrirFecharBlt);
379 
380 
381     Border bords_robot= BorderFactory.createLineBorder(new Color
    (0,0,0),1);
382     TitledBorder borda_robot = BorderFactory.createTitledBorder(
    bords_robot, "Robot");
383     JPanel panel = new JPanel();
384     panel.setName("Canal de Comunicação");
385     panel.setBorder(new LineBorder(new Color(0, 0, 0)));
386     panel.setBounds(10, 34, 719, 66);
387     panel.setBorder(borda_robot);
388     getContentPane().add(panel);
389     contentPane = new JPanel();
390     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
391 
392     addWindowListener(new WindowAdapter(){
393             public void windowClosing(WindowEvent e){
394                 if (bd.isLigado())
395                 {
396                     txtLog.append(" Desconectando o robot ... \n");
397                     System.out.println("desconect");
398                     bd.getRobot().CloseEV3();
399                     rdbtnAbrirFecharBlt.setSelected(false);
400                 }
```

```
                else {
                    System.out.println("Closing program");
                    System.exit(0);
                }
            }
        });

    setVisible(true);
  }

}

Classe App_Rei

package ptrabalho;

public class App_Rei
{
    @SuppressWarnings("unused")
    private GUI_Rei gui;
    private BD_Rei bd;
    Mensagem msg = null;
    private int state = 2;
    private int counter = 0;
    private final int escreverMensagem = 1;
    private final int dormir = 2;
    private final int esperarTempoExecucao = 3;

    public App_Rei(String[] args)
    {
        bd = new BD_Rei();

        if (args.length == 2 )
        {
          int nMensagens = Integer.parseInt(args[0]);
          if (nMensagens >= 8 && nMensagens <= 12)
            bd.setNMensagens(nMensagens);
          bd.setFile(args[1]);
        }
        gui = new GUI_Rei(bd);
    }

    public BD_Rei getBD()
    {
        return bd;
    }

    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 = dormir;
                break;
              }
              else {
                state = dormir;
                break;
            }

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

      }
    }
    System.out.println("sai");
        Thread.sleep(100);

    }

    public static void main(String[] args) throws
    InterruptedException
    {

        App_Rei app = new App_Rei(args);
        System.out.println("A aplicação começou.");
        app.run();
        System.out.println("A aplicação terminou.");
    }

}


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();
```

```
518         terminar = false;
519         ligado = false;
520 
521     }
522 
523 
524     public boolean getTerminar()
525     {
526         return terminar;
527     }
528 
529     public void setTerminar(boolean b)
530     {
531         terminar = b;
532     }
533 
534     public boolean isLigado()
535     {
536         return ligado;
537     }
538 
539     public void setLigado(boolean b)
540     {
541         ligado = b;
542     }
543 
544     public void setNome(String n)
545     {
546         nome = n;
547     }
548 
549     public String getNome()
550     {
551         return nome;
552     }
553 
554 }
555 
556 
557 package ptrabalho;
558 
559 import java.util.ArrayList;
560 import java.util.List;
561 
562 public class BD_Base {
563 
564   protected int distance;
565   protected int angulo;
566     protected int raio;
567     protected String file = "";
568     protected CanalComunicacaoConsistente ccc = null;
569     protected int nMensagens = 8;
570     private List<Mensagem> mensagens = new ArrayList<>();
571 
572   public BD_Base() {
573     distance = 30;
574         angulo = 90;
575         raio = 20;
576   }
577 
```

```
578   public int getDist()
579     {
580         return distance;
581     }
582 
583     public void setDist(int i)
584     {
585         distance = i;
586     }
587 
588     public int getAng()
589     {
590         return angulo;
591     }
592 
593     public void setAng(int i)
594     {
595         angulo = i;
596     }
597 
598     public int getRaio()
599     {
600         return raio;
601     }
602 
603     public void setRaio(int i)
604     {
605         raio = i;
606     }
607 
608     public String getFile()
609     {
610         return file;
611     }
612 
613     public void setFile(String f)
614     {
615         file = f;
616     }
617 
618     public CanalComunicacaoConsistente getCanal()
619     {
620         return ccc;
621     }
622 
623     public void setCanal(CanalComunicacaoConsistente c)
624     {
625         ccc = c;
626     }
627 
628     public int getNMensagens()
629     {
630         return nMensagens;
631     }
632 
633     public void setNMensagens(int i)
634     {
635       nMensagens = i;
636     }
637 
```

```
638     public void addMensagem(Mensagem msg)
639     {
640       mensagens.add(msg);
641     }
642 
643     public void removeMensagem()
644     {
645       mensagens.remove(0);
646     }
647 
648     public List<Mensagem> getMensagens()
649     {
650     return mensagens;
651     }
652 
653 
654 }
655 
656 
657 
658 package ptrabalho;
659 
660 import java.awt.Color;
661 import java.awt.EventQueue;
662 import java.awt.Font;
663 import java.awt.event.ActionEvent;
664 import java.awt.event.ActionListener;
665 import java.util.Random;
666 
667 import javax.swing.BorderFactory;
668 import javax.swing.JButton;
669 import javax.swing.JFrame;
670 import javax.swing.JPanel;
671 import javax.swing.border.Border;
672 import javax.swing.border.EmptyBorder;
673 import javax.swing.border.LineBorder;
674 import javax.swing.border.TitledBorder;
675 
676 
677 
678 
679 public class GUI_Rei extends GUI_Base
680 {
681   private int id = 0;
682 
683   private JPanel contentPane;
684 
685   private int state = 0;
686   private final int gerarRandom = 0;
687   private final int escreverMensagem = 1;
688   private final int dormir = 2;
689   private final int esperarTempoExecucao = 3;
690 
691   Mensagem msg = null;
692 
693   /**
694    * Launch the application.
695    */
696   /*public static void main(String[] args)
697   {
```

```
698     EventQueue.invokeLater(new Runnable()
699     {
700       public void run() {
701         try
702         {
703           GUI_Rei frame = new GUI_Rei(bd);
704           frame.setVisible(true);
705         } catch (Exception e)
706         {
707           e.printStackTrace();
708         }
709       }
710     });
711   }*/
712
713   public GUI_Rei(BD_Rei bd)
714   {
715     super(bd);
716     EventQueue.invokeLater(new Runnable()
717     {
718       public void run()
719       {
720         try
721         {
722           init_GUI_Rei(bd);
723
724         } catch (Exception e)
725         {
726           e.printStackTrace();
727         }
728       }
729     });
730   }
731
732   /**
733    * Create the frame.
734    */
735   public void init_GUI_Rei(BD_Rei bd)
736   {
737     setTitle("Trabalho 1 - GUI Rei");
738     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
739     setBounds(100, 100, 754, 697);
740
741     Border cost = BorderFactory.createLineBorder(new Color(0,0,0)
      ,1);
742     TitledBorder borda_rei = BorderFactory.createTitledBorder(cost,
      "Controle do Robot em Modo Automático");
743     JPanel panel_1_1 = new JPanel();
744     panel_1_1.setLayout(null);
745     panel_1_1.setName("Controle do Robot em Modo Automático");
746     panel_1_1.setBorder(new LineBorder(new Color(0, 0, 0)));
747     panel_1_1.setBounds(10, 34, 719, 66);
748     panel_1_1.setBorder(borda_rei);
749     getContentPane().add(panel_1_1);
750
751     JButton btn8com = new JButton("8 Comandos Aleatórios");
752     btn8com.setEnabled(false);
753     btn8com.setFont(new Font("Arial", Font.BOLD, 12));
754     btn8com.setBounds(10, 26, 300, 25);
755     panel_1_1.add(btn8com);
```

```
756
757     JButton btn16com = new JButton("16 Comandos Aleatórios");
758     btn16com.setEnabled(false);
759     btn16com.setFont(new Font("Arial", Font.BOLD, 12));
760     btn16com.setBounds(385, 26, 300, 25);
761     panel_1_1.add(btn16com);
762     contentPane = new JPanel();
763     contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
764
765
766
767     rdbtnAtivarDesativarComp.addActionListener(new ActionListener()
768     {
769       public void actionPerformed(ActionEvent e)
770       {
771         if(!rdbtnAtivarDesativarComp.isSelected())
772         {
773           btn8com.setEnabled(false);
774           btn16com.setEnabled(false);
775         }
776         else {
777           btn8com.setEnabled(true);
778           btn16com.setEnabled(true);
779         }
780       }
781     });
782
783     btnFrt.addActionListener(new ActionListener()
784     {
785       public void actionPerformed(ActionEvent e)
786       {
787         Mensagem mensagem = new Mensagem(id,1,bd.getDist(),0);
788         bd.addMensagem(mensagem);
789       }
790     });
791
792     btnEsq.addActionListener(new ActionListener()
793     {
794       public void actionPerformed(ActionEvent e)
795       {
796         Mensagem mensagem = new Mensagem(id,3,bd.getRaio(),bd.
    getAng());
797         bd.addMensagem(mensagem);
798       }
799     });
800
801     btnDir.addActionListener(new ActionListener()
802     {
803       public void actionPerformed(ActionEvent e)
804       {
805         Mensagem mensagem = new Mensagem(id,2,bd.getRaio(),bd.
    getAng());
806         bd.addMensagem(mensagem);
807       }
808     });
809
810     btnTras.addActionListener(new ActionListener()
811     {
812       public void actionPerformed(ActionEvent e)
813       {
```

```
814         Mensagem mensagem = new Mensagem(id,1,-bd.getDist(),0);
815         bd.addMensagem(mensagem);
816       }
817     });
818 
819     btnParar.addActionListener(new ActionListener()
820     {
821       public void actionPerformed(ActionEvent e)
822       {
823         Mensagem mensagem = new Mensagem(id,0,0,0);
824         bd.addMensagem(mensagem);
825       }
826     });
827 
828     btn8com.addActionListener(new ActionListener()
829     {
830       public void actionPerformed(ActionEvent e)
831       {
832         for (int i = 0; i<8; i++)
833         {
834           msg = gerarRandomMensagem();
835           bd.addMensagem(msg);
836 
837         }
838       }
839     });
840 
841     btn16com.addActionListener(new ActionListener()
842     {
843       public void actionPerformed(ActionEvent e)
844       {
845         for (int i = 0; i<16; i++)
846         {
847           msg = gerarRandomMensagem();
848           bd.addMensagem(msg);
849         }
850 
851       }
852     });
853 
854     rdbtnAbrirCanal.addActionListener(new ActionListener()
855     {
856       public void actionPerformed(ActionEvent e)
857       {
858         if (rdbtnAbrirCanal.isSelected())
859         {
860           CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente(bd.getNMensagens());
861           cc.abrirCanalEscritor(bd.getFile());
862           bd.setCanal(cc);
863           txtLog.append(" Canal de Comunicação " + bd.getFile() + "
    aberto!! \n");
864             }
865         else {
866           bd.getCanal().fecharCanal();
867           bd.setCanal(null);
868           txtLog.append(" Canal de Comunicação " + bd.getFile() + "
    fechado \n");
869         }
870       }
```

```
871       });
872
873
874
875       setVisible(true);
876     }
877
878
879
880     private Mensagem gerarRandomMensagem() {
881       Mensagem m = new Mensagem();
882       Random rn = new Random();
883       int[] variaveis = new int[4]; // array de 4 variaveis
884       int tipoMensagem = rn.nextInt(3); // random between 0-2
885       if(id==8)
886         id=0;
887       variaveis = gerarVariaveis(tipoMensagem);
888       m.setId(variaveis[0]);
889       m.setTipo(variaveis[1]);
890       m.setArg1(variaveis[2]);
891       m.setArg2(variaveis[3]);
892       return m;
893     }
894
895     /*
896      * Metodo auxiliar ao gerarRandomMensagem()
897      *
898      * @param tMsg - tipo de mensagem
899      */
900     private int[] gerarVariaveis(int tMsg) {
901       Random rn = new Random();
902       int[] variaveis = new int[4];
903       int variavel;
904       if (tMsg == 0) { // para tMsg 0 faz reta
905         variaveis[0] = id;
906         variaveis[1] = 1; // 1 equivale a reta na minha mensagem
907         variavel = rn.nextInt(45) + 5; // random between 5-50 cm reta
908         variaveis[2] = variavel;
909         variaveis[3] = 0;
910       } else if (tMsg == 1) { // para tMsg 1 faz curva direita
911         variaveis[0] = id;
912         variaveis[1] = 2; // 2 equivale a reta na minha mensagem
913         variavel = rn.nextInt(30); // random between 0-30 raio
914         variaveis[2] = variavel;
915         variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
916         variaveis[3] = variavel;
917       } else { // para tMsg 0 faz curva esquerda
918         variaveis[0] = id;
919         variaveis[1] = 3; // 3 equivale a reta na minha mensagem
920         variavel = rn.nextInt(30); // random between 0-30 raio
921         variaveis[2] = variavel;
922         variavel = rn.nextInt(70) + 20;// random between 20-90 angulo
923         variaveis[3] = variavel;
924       }
925       return variaveis;
926     }
927 }
928
929
930
```

```
931 package ptrabalho;
932
933 import java.awt.EventQueue;
934
935 import javax.swing.JFrame;
936 import javax.swing.JPanel;
937 import javax.swing.border.Border;
938 import javax.swing.border.EmptyBorder;
939 import javax.swing.BoxLayout;
940 import java.awt.GridLayout;
941
942 import javax.swing.BorderFactory;
943 import javax.swing.Box;
944 import java.awt.CardLayout;
945 import javax.swing.JTextField;
946 import java.awt.FlowLayout;
947 import javax.swing.JLabel;
948 import javax.swing.SwingConstants;
949 import java.awt.Font;
950 import javax.swing.JButton;
951 import javax.swing.JToggleButton;
952 import javax.swing.JSpinner;
953 import javax.swing.JRadioButton;
954 import javax.swing.JSeparator;
955 import javax.swing.JComboBox;
956 import javax.swing.JFileChooser;
957 import javax.swing.JTextPane;
958 import java.awt.Panel;
959 import java.awt.Color;
960 import java.awt.Canvas;
961 import javax.swing.border.LineBorder;
962 import javax.swing.border.TitledBorder;
963 import javax.swing.event.ChangeEvent;
964 import javax.swing.event.ChangeListener;
965 import javax.swing.JCheckBox;
966 import java.awt.event.ActionListener;
967 import java.io.File;
968 import java.awt.event.ActionEvent;
969 import javax.swing.JTextArea;
970 import javax.swing.JScrollPane;
971 import javax.swing.SpinnerNumberModel;
972
973 public class GUI_Base extends JFrame
974 {
975
976   private JPanel contentPane;
977   protected JTextField txtRaio;
978   protected JTextField txtAng;
979   protected JTextField txtDist;
980   protected JTextField txtFile;
981   protected JButton btnFrt;
982   protected JButton btnEsq;
983   protected JButton btnDir;
984   protected JButton btnParar;
985   protected JButton btnTras;
986   protected JTextArea txtLog;
987   protected JButton btnFile;
988   protected JButton btnLimpaLog;
989   protected JCheckBox rdbtnAtivarDesativarComp;
990   protected JRadioButton rdbtnAbrirCanal;
```

```
991   protected JSpinner spinNMensagem;
992
993
994   /**
995    * Launch the application.
996    */
997   /*public static void main(String[] args)
998   {
999     EventQueue.invokeLater(new Runnable()
1000    {
1001      public void run()
1002      {
1003        try
1004        {
1005          GUI_Base frame = new GUI_Base();
1006          frame.setVisible(true);
1007        } catch (Exception e)
1008        {
1009          e.printStackTrace();
1010        }
1011      }
1012    });
1013  }*/
1014
1015  /**
1016   * Create the frame.
1017   */
1018  public GUI_Base(BD_Base bd)
1019  {
1020    setTitle("Trabalho 1 - GUI Base");
1021    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
1022    setBounds(100, 100, 754, 691);
1023    contentPane = new JPanel();
1024    contentPane.setBorder(new EmptyBorder(100, 100,100, 100));
1025
1026    setContentPane(contentPane);
1027    contentPane.setLayout(null);
1028
1029    txtFile = new JTextField();
1030    txtFile.setFont(new Font("Arial", Font.BOLD, 12));
1031    txtFile.setBounds(143, 156, 494, 25);
1032    contentPane.add(txtFile);
1033    txtFile.setColumns(10);
1034
1035    JLabel lblNewLabel = new JLabel("Ficheiro do Canal");
1036    lblNewLabel.setFont(new Font("Arial", Font.BOLD, 12));
1037    lblNewLabel.setHorizontalAlignment(SwingConstants.LEFT);
1038    lblNewLabel.setBounds(30, 156, 103, 25);
1039    contentPane.add(lblNewLabel);
1040
1041    btnFile = new JButton("...");
1042    btnFile.setFont(new Font("Arial", Font.BOLD, 12));
1043    btnFile.setBounds(647, 157, 61, 25);
1044    contentPane.add(btnFile);
1045
1046    JLabel lblNMsg = new JLabel("N   msg");
1047    lblNMsg.setHorizontalAlignment(SwingConstants.LEFT);
1048    lblNMsg.setFont(new Font("Arial", Font.BOLD, 12));
1049    lblNMsg.setBounds(30, 200, 103, 25);
1050    contentPane.add(lblNMsg);
```

```
1051
1052     spinNMensagem = new JSpinner();
1053     spinNMensagem.setModel(new SpinnerNumberModel(8, 8, 12, 1));
1054     spinNMensagem.setFont(new Font("Arial", Font.BOLD, 12));
1055     spinNMensagem.setBounds(143, 200, 50, 25);
1056     contentPane.add(spinNMensagem);
1057
1058     rdbtnAbrirCanal = new JRadioButton("Abrir/Fechar Canal");
1059     rdbtnAbrirCanal.setFont(new Font("Arial", Font.BOLD, 12));
1060     rdbtnAbrirCanal.setBounds(459, 200, 103, 25);
1061     contentPane.add(rdbtnAbrirCanal);
1062
1063     Border simpb = BorderFactory.createLineBorder(new Color(0,0,0)
    ,1);
1064     TitledBorder border_simp = BorderFactory.createTitledBorder(
    simpb,"Canal de Comunicação");
1065     JPanel panel = new JPanel();
1066     panel.setName("Canal de Comunicação");
1067     panel.setBorder(new LineBorder(new Color(0, 0, 0)));
1068     panel.setBounds(10, 128, 719, 113);
1069     panel.setBorder(border_simp);
1070     contentPane.add(panel);
1071
1072     JLabel lblRaio = new JLabel("Raio");
1073     lblRaio.setHorizontalAlignment(SwingConstants.LEFT);
1074     lblRaio.setFont(new Font("Arial", Font.BOLD, 12));
1075     lblRaio.setBounds(30, 270, 103, 25);
1076     contentPane.add(lblRaio);
1077
1078     txtRaio = new JTextField();
1079     txtRaio.setText("20");
1080
1081     txtRaio.setFont(new Font("Arial", Font.BOLD, 12));
1082     txtRaio.setColumns(10);
1083     txtRaio.setBounds(143, 270, 100, 23);
1084     contentPane.add(txtRaio);
1085
1086     JLabel lblngulo = new JLabel("Ângulo");
1087     lblngulo.setHorizontalAlignment(SwingConstants.LEFT);
1088     lblngulo.setFont(new Font("Arial", Font.BOLD, 12));
1089     lblngulo.setBounds(30, 300, 103, 25);
1090     contentPane.add(lblngulo);
1091
1092     JLabel lblDistncia = new JLabel("Distância");
1093     lblDistncia.setHorizontalAlignment(SwingConstants.LEFT);
1094     lblDistncia.setFont(new Font("Arial", Font.BOLD, 12));
1095     lblDistncia.setBounds(30, 330, 103, 25);
1096     contentPane.add(lblDistncia);
1097
1098     txtAng = new JTextField();
1099     txtAng.setText("90");
1100     txtAng.setFont(new Font("Arial", Font.BOLD, 12));
1101     txtAng.setColumns(10);
1102     txtAng.setBounds(143, 300, 100, 23);
1103     contentPane.add(txtAng);
1104
1105     txtDist = new JTextField();
1106     txtDist.setText("30");
1107     txtDist.setFont(new Font("Arial", Font.BOLD, 12));
1108     txtDist.setColumns(10);
```

```
1109     txtDist.setBounds(143, 330, 100, 23);
1110     contentPane.add(txtDist);
1111 
1112     btnParar = new JButton("Parar");
1113     btnParar.setEnabled(false);
1114     btnParar.setFont(new Font("Arial", Font.BOLD, 12));
1115     btnParar.setBounds(470, 297, 100, 25);
1116     contentPane.add(btnParar);
1117 
1118     btnFrt = new JButton("Frente");
1119     btnFrt.setEnabled(false);
1120     btnFrt.setFont(new Font("Arial", Font.BOLD, 12));
1121     btnFrt.setBounds(470, 269, 100, 25);
1122     contentPane.add(btnFrt);
1123 
1124     btnDir = new JButton("Direita");
1125     btnDir.setEnabled(false);
1126     btnDir.setFont(new Font("Arial", Font.BOLD, 12));
1127     btnDir.setBounds(575, 298, 100, 25);
1128     contentPane.add(btnDir);
1129 
1130     btnEsq = new JButton("Esquerda");
1131     btnEsq.setEnabled(false);
1132     btnEsq.setFont(new Font("Arial", Font.BOLD, 12));
1133     btnEsq.setBounds(365, 297, 100, 25);
1134     contentPane.add(btnEsq);
1135 
1136     btnTras = new JButton("Tras");
1137     btnTras.setEnabled(false);
1138     btnTras.setFont(new Font("Arial", Font.BOLD, 12));
1139     btnTras.setBounds(470, 325, 100, 25);
1140     contentPane.add(btnTras);
1141 
1142     Border borda_cont_robot = BorderFactory.createLineBorder(new
    Color(0,0,0),1);
1143     TitledBorder borda1 = BorderFactory.createTitledBorder(
    borda_cont_robot,"Controle do Robot");
1144     JPanel panel_1 = new JPanel();
1145     panel_1.setFont(new Font("Arial", Font.BOLD, 12));
1146     panel_1.setName("Controle do Robot");
1147     panel_1.setBorder(new LineBorder(new Color(0, 0, 0)));
1148     panel_1.setBounds(10, 251, 719, 113);
1149     panel_1.setBorder(borda1);
1150     contentPane.add(panel_1);
1151 
1152     rdbtnAtivarDesativarComp = new JCheckBox("Ativar / Desativar
    Comportamento");
1153     rdbtnAtivarDesativarComp.setFont(new Font("Arial", Font.BOLD,
    12));
1154     rdbtnAtivarDesativarComp.setBounds(10, 365, 262, 45);
1155     contentPane.add(rdbtnAtivarDesativarComp);
1156 
1157     JLabel lblLog = new JLabel("Log");
1158     lblLog.setHorizontalAlignment(SwingConstants.LEFT);
1159     lblLog.setFont(new Font("Arial", Font.BOLD, 12));
1160     lblLog.setBounds(10, 429, 103, 25);
1161     contentPane.add(lblLog);
1162 
1163     btnLimpaLog = new JButton("Limpar Log");
1164     btnLimpaLog.setFont(new Font("Arial", Font.BOLD, 12));
```

```
1165      btnLimpaLog.setBounds(10, 630, 719, 25);
1166      contentPane.add(btnLimpaLog);
1167
1168
1169      txtLog = new JTextArea();
1170      txtLog.setBounds(45, 464, 659, 156);
1171      contentPane.add(txtLog);
1172
1173      JScrollPane scrollPane = new JScrollPane(txtLog);
1174      scrollPane.setBounds(45, 464, 663, 156);
1175      contentPane.add(scrollPane);
1176
1177      btnLimpaLog.addActionListener(new ActionListener()
1178      {
1179        public void actionPerformed(ActionEvent e)
1180        {
1181          txtLog.setText("");
1182        }
1183      });
1184
1185      spinNMensagem.addChangeListener(new ChangeListener() {
1186        @Override
1187        public void stateChanged(ChangeEvent e) {
1188          int nMensagens = (int) spinNMensagem.getValue();
1189          if (bd.getCanal()!= null) {
1190
1191            bd.getCanal().nMensagens = nMensagens;
1192            bd.getCanal().BUFFER_MAX = 16 * nMensagens;
1193
1194          }
1195          bd.setNMensagens(nMensagens);
1196          txtLog.append(" O num de mensagens é " + nMensagens + "\n")
      ;
1197        }
1198      });
1199
1200      btnFile.addActionListener(new ActionListener()
1201      {
1202        public void actionPerformed(ActionEvent e)
1203        {
1204          txtLog.append(" Escolher Ficheiro \n");
1205          JFileChooser fileChooser = new JFileChooser(System.
      getProperty("user.dir"));
1206
1207          if (fileChooser.showSaveDialog(null) == JFileChooser.
      APPROVE_OPTION) {
1208            String file = fileChooser.getSelectedFile().
      getAbsolutePath();
1209            txtFile.setText(file);
1210            bd.setFile(file);
1211            txtLog.append(" O nome do ficheiro é " + bd.getFile() + "
      \n");
1212          }
1213        }
1214      });
1215
1216      txtFile.addActionListener(new ActionListener()
1217      {
1218        public void actionPerformed(ActionEvent e)
1219        {
```

```
1220        bd.setFile(txtFile.getText());
1221        txtLog.append(" O nome do ficheiro é " + bd.getFile() + "\n
    ");
1222      }
1223    });
1224
1225
1226    rdbtnAtivarDesativarComp.addActionListener(new ActionListener()
1227    {
1228      public void actionPerformed(ActionEvent e)
1229      {
1230        if(!rdbtnAtivarDesativarComp.isSelected())
1231        {
1232          btnFrt.setEnabled(false);
1233          btnEsq.setEnabled(false);
1234          btnParar.setEnabled(false);
1235          btnDir.setEnabled(false);
1236          btnTras.setEnabled(false);
1237        }
1238        else {
1239          btnFrt.setEnabled(true);
1240          btnEsq.setEnabled(true);
1241          btnParar.setEnabled(true);
1242          btnDir.setEnabled(true);
1243          btnTras.setEnabled(true);
1244        }
1245      }
1246    });
1247
1248
1249    spinNMensagem.setValue(bd.getNMensagens());
1250    txtFile.setText(bd.getFile());
1251
1252    txtDist.addActionListener(new ActionListener()
1253    {
1254      public void actionPerformed(ActionEvent e)
1255      {
1256        bd.setDist(Integer.parseInt(txtDist.getText()));
1257        txtLog.append(" A distancia é " + bd.getDist() + "\n");
1258      }
1259    });
1260
1261    txtAng.addActionListener(new ActionListener()
1262    {
1263      public void actionPerformed(ActionEvent e)
1264      {
1265        bd.setAng(Integer.parseInt(txtAng.getText()));
1266        txtLog.append(" O angulo é " + bd.getAng() + "\n");
1267      }
1268    });
1269
1270    txtRaio.addActionListener(new ActionListener()
1271    {
1272      public void actionPerformed(ActionEvent e)
1273      {
1274        bd.setRaio(Integer.parseInt(txtRaio.getText()));
1275        txtLog.append(" A raio é " + bd.getRaio() + "\n");
1276      }
1277    });
1278
```

```
1279   }
1280 }
1281
1282 Classe CanalComunicacaoConsistente
1283
1284 package ptrabalho;
1285
1286 public class CanalComunicacaoConsistente extends CanalComunicacao{
1287
1288
1289     public CanalComunicacaoConsistente(int n) {
1290     super(n);
1291     // TODO Auto-generated constructor stub
1292   }
1293
1294   public boolean GetandSetWrite(Mensagem msg) {
1295         Mensagem mensagem = GetandSetRead();
1296         //System.out.println(mensagem);
1297         try {
1298           if(mensagem == null)
1299             return false;
1300             if(mensagem.tipo == iMensagem.vazia) {
1301                 System.out.println("entrou, no buffer: " + mensagem
     + "enviei msg = " + msg);
1302
1303                 fl = canal.lock();
1304
1305                 enviarMensagem(msg, true);
1306
1307                 fl.release();
1308                 return true;
1309             }else {
1310               //System.out.println("NÃO entrou, no buffer: " +
    mensagem + "quero enviar msg = " + msg);
1311             }
1312         } catch (Exception e) {
1313             e.printStackTrace();
1314         }
1315
1316         return false;
1317     }
1318
1319     public Mensagem GetandSetRead(){
1320
1321         try {
1322             fl = canal.lock();
1323
1324             Mensagem mensagem = receberMensagem(true);
1325
1326             //limparLida();
1327
1328             fl.release();
1329
1330             return mensagem;
1331
1332         } catch (Exception e) {
1333             e.printStackTrace();
1334         }
1335
1336         return null;
```

```
1337     }
1338 
1339     public Mensagem GetandSetReadLeitor(){
1340 
1341         try {
1342             fl = canal.lock();
1343 
1344             Mensagem mensagem = receberMensagem(false);
1345             System.out.println(mensagem);
1346 
1347             limparLida();
1348 
1349             fl.release();
1350 
1351             return mensagem;
1352 
1353         } catch (Exception e) {
1354             e.printStackTrace();
1355         }
1356 
1357         return null;
1358     }
1359 
1360 
1361 
1362     /*public static void main (String[] args) {
1363         CanalComunicacaoConsistente cc = new
    CanalComunicacaoConsistente();
1364 
1365         MensagemParar msgParar = new MensagemParar(0, 0, false);
    //int id, int tipo, boolean sincrono
1366         MensagemReta msg     = new MensagemReta(1,1,20);          //
    int id, int tipo, int dist
1367         MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15);
    //int id, int tipo, int raio, int ang
1368         MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);
1369 
1370         //Testar
1371         cc.abrirCanal("comunicacao.dat");
1372 
1373 
1374         cc.GetandSetWrite(msg);
1375         System.out.println(cc.receberMensagem());
1376         cc.GetandSetWrite(msgCD);
1377         cc.GetandSetWrite(msgCD);
1378         cc.GetandSetWrite(msgCE);
1379         cc.GetandSetWrite(msgCD);
1380         cc.GetandSetWrite(msgCD);
1381         cc.GetandSetWrite(msgParar);
1382         cc.GetandSetWrite(msgParar);
1383         cc.GetandSetWrite(msgCD);
1384         cc.GetandSetWrite(msgCD);
1385         cc.GetandSetWrite(msgCD);
1386         cc.GetandSetWrite(msgCD);
1387         cc.GetandSetWrite(msgParar);
1388         cc.GetandSetWrite(msgParar);
1389         cc.GetandSetWrite(msgCD);
1390         cc.GetandSetWrite(msgCD);
1391         cc.GetandSetWrite(msgCD);
1392         cc.GetandSetWrite(msgParar);
```

```
1393            cc.GetandSetWrite(msgParar);
1394
1395            cc.fecharCanal();
1396        }*/
1397
1398    }
1399
1400    Classe CanalComunicacao
1401
1402    package ptrabalho;
1403    import java.io.*;
1404    import java.nio.MappedByteBuffer;
1405    import java.nio.channels.FileChannel;
1406    import java.nio.channels.FileLock;
1407
1408    public class CanalComunicacao {
1409        FileLock fl;
1410
1411        // ficheiro
1412        private File ficheiro;
1413
1414        // canal que liga o conteúdo do ficheiro ao Buffer
1415        protected FileChannel canal;
1416
1417        // buffer
1418        private MappedByteBuffer buffer;
1419
1420        // dimensão máxima em bytes do buffer = (N   grup + 1) * 4
       bytes (int) * 4 ints
1421        //final int nGrupo = 15;
1422        //final int BUFFER_MAX = (nGrupo + 1) * 4 * 4;
1423        protected int nMensagens = 8;
1424        protected int BUFFER_MAX = nMensagens*16;
1425        protected int putBuffer, getBuffer;
1426
1427
1428
1429        //Construtor onde se cria o canal
1430        public CanalComunicacao(int n) {
1431            ficheiro = null;
1432            canal    = null;
1433            buffer   = null;
1434            nMensagens = n;
1435            BUFFER_MAX = nMensagens*16;
1436        }
1437
1438        public boolean abrirCanalEscritor(String Filename) {
1439            // cria um ficheiro
1440            ficheiro = new File(Filename);
1441
1442            //cria um canal de comunicação de leitura e escrita
1443            try {
1444                canal = new RandomAccessFile(ficheiro, "rw").getChannel
       ();
1445            } catch (FileNotFoundException e) {
1446                return false;
1447            }
1448
1449            // mapeia para memória o conteúdo do ficheiro
1450            try {
```

```
1451            buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
    BUFFER_MAX);
1452
1453            //Inicializar apontadores - put e get buffer a zero
1454            getBuffer = 0;
1455            putBuffer = 0;
1456
1457            //Inicializar buffer - dizer que todos os blocos são
    vazios
1458            inicializarBuffer();
1459
1460
1461        } catch (IOException e) {
1462            return false;
1463        }
1464
1465        return true;
1466    }
1467
1468    public boolean abrirCanalLeitor(String Filename) {
1469        // cria um ficheiro
1470        ficheiro = new File(Filename);
1471
1472        //cria um canal de comunicação de leitura e escrita
1473        try {
1474            canal = new RandomAccessFile(ficheiro, "rw").getChannel
    ();
1475        } catch (FileNotFoundException e) {
1476            return false;
1477        }
1478
1479        // mapeia para memória o conteúdo do ficheiro
1480        try {
1481            buffer = canal.map(FileChannel.MapMode.READ_WRITE, 0,
    BUFFER_MAX);
1482
1483            //Inicializar apontadores - put e get buffer a zero
1484            getBuffer = 0;
1485            putBuffer = 0;
1486
1487
1488
1489
1490        } catch (IOException e) {
1491            return false;
1492        }
1493
1494        return true;
1495    }
1496
1497    // recebe e converte numa Mensagem, lê e retorna mensagem
1498    //true escritor// false leitor
1499    Mensagem receberMensagem(boolean e_l) {
1500
1501        int id, tipo, distancia, raio, angulo;
1502        boolean sincrono;
1503
1504        buffer.asIntBuffer();
1505        //buffer.position(0);
```

```
1506        //System.out.println("getbuffer" + getBuffer + "nMensanges"
    + nMensagens);
1507        buffer.position(getBuffer * 16);
1508
1509        id   = buffer.getInt();
1510        tipo = buffer.getInt();
1511
1512        Mensagem msg = null;
1513
1514        switch (tipo) {
1515            case iMensagem.parar:
1516                sincrono = buffer.getInt() == 1;
1517                msg = new MensagemParar(id, tipo, sincrono);
1518                break;
1519
1520            case iMensagem.reta:
1521                distancia = buffer.getInt();
1522                msg = new MensagemReta(id, tipo, distancia);
1523                break;
1524
1525            case iMensagem.curvarDir:
1526                raio = buffer.getInt();
1527                angulo = buffer.getInt();
1528                msg = new MensagemCurvar(id, tipo, raio, angulo);
1529                break;
1530
1531            case iMensagem.curvarEsq:
1532                raio = buffer.getInt();
1533                angulo = buffer.getInt();
1534                msg = new MensagemCurvar(id, tipo, raio, angulo);
1535                break;
1536
1537            case iMensagem.vazia:
1538                msg = new Mensagem(id, tipo, 0, 0);
1539                break;
1540        }
1541
1542        if(e_l) {
1543          if(tipo==4)
1544            getBuffer =++ getBuffer % nMensagens;
1545        }
1546        else {
1547          if (tipo != 4)
1548              getBuffer =++ getBuffer % nMensagens;
1549        }
1550
1551        return msg;
1552
1553    }
1554
1555 // recebe e converte numa Mensagem, lê e retorna mensagem
1556    Mensagem receberMensagemLeitor() {
1557
1558        int id, tipo, distancia, raio, angulo;
1559        boolean sincrono;
1560
1561        buffer.asIntBuffer();
1562        //buffer.position(0);
1563        //System.out.println("getbuffer" + getBuffer + "nMensanges"
    + nMensagens);
```

```
1564        buffer.position(getBuffer * 16);
1565
1566        id   = buffer.getInt();
1567        tipo = buffer.getInt();
1568
1569        Mensagem msg = null;
1570
1571        switch (tipo) {
1572            case iMensagem.parar:
1573                sincrono = buffer.getInt() == 1;
1574                msg = new MensagemParar(id, tipo, sincrono);
1575                break;
1576
1577            case iMensagem.reta:
1578                distancia = buffer.getInt();
1579                msg = new MensagemReta(id, tipo, distancia);
1580                break;
1581
1582            case iMensagem.curvarDir:
1583                raio = buffer.getInt();
1584                angulo = buffer.getInt();
1585                msg = new MensagemCurvar(id, tipo, raio, angulo);
1586                break;
1587
1588            case iMensagem.curvarEsq:
1589                raio = buffer.getInt();
1590                angulo = buffer.getInt();
1591                msg = new MensagemCurvar(id, tipo, raio, angulo);
1592                break;
1593
1594            case iMensagem.vazia:
1595                msg = new Mensagem(id, tipo, 0, 0);
1596                break;
1597        }
1598
1599        if (tipo != 4)
1600          getBuffer =++ getBuffer % nMensagens;
1601
1602        return msg;
1603
1604    }
1605
1606    // envia uma Mensagem como um conjunto de ints
1607    void enviarMensagem(Mensagem msg, boolean e_c) {
1608
1609        try {
1610            //buffer.position(0);
1611            buffer.position(putBuffer * 16);
1612
1613            // Obter ID e escrevê-lo no buffer
1614            int id = msg.getId();
1615            buffer.putInt(id);
1616
1617            // Obter Tipo e escrevê-lo no buffer
1618            int tipo = msg.getTipo();
1619            buffer.putInt(tipo);
1620
1621            // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
1622            switch (tipo) {
```

```
1623                case iMensagem.parar:
1624                    buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
1625                    break;
1626
1627                case iMensagem.reta:
1628                    buffer.putInt(msg.getArg1());
1629                    break;
1630
1631                case iMensagem.curvarDir:
1632                  System.out.println("yo");
1633                    buffer.putInt(msg.getArg1());
1634                    buffer.putInt(msg.getArg2());
1635                    break;
1636
1637                case iMensagem.curvarEsq:
1638                    buffer.putInt(msg.getArg1());
1639                    buffer.putInt(msg.getArg2());
1640                    break;
1641
1642                case iMensagem.vazia:
1643                    buffer.putInt(0);
1644                    System.out.println("limpei");
1645                    //id = -1;
1646                    break;
1647            }
1648
1649            if (e_c) {
1650              if (tipo != 4)
1651                  putBuffer =++ putBuffer % nMensagens;
1652            }
1653            else
1654            {
1655              putBuffer =++ putBuffer % nMensagens;
1656                msg.setId(id + 1);
1657            }
1658
1659            //msg.setId(id + 1);
1660
1661        }catch(Exception e) {
1662            e.printStackTrace();
1663        }
1664
1665    }
1666
1667    void enviarMensagemClean(Mensagem msg) {
1668
1669        try {
1670            //buffer.position(0);
1671            buffer.position(putBuffer * 16);
1672
1673            // Obter ID e escrevê-lo no buffer
1674            int id = msg.getId();
1675            buffer.putInt(id);
1676
1677            // Obter Tipo e escrevê-lo no buffer
1678            int tipo = msg.getTipo();
1679            buffer.putInt(tipo);
1680
```

```
1681            // Escrever o conteúdo no buffer, consoante o tipo da
    mensagem (Ex: para o tipo reta -> escrever distância
1682            switch (tipo) {
1683                case iMensagem.parar:
1684                    buffer.putInt(((MensagemParar) msg).isSincrono
    () ? 1 : 0);
1685                    break;
1686
1687                case iMensagem.reta:
1688                    buffer.putInt(msg.getArg1());
1689                    break;
1690
1691                case iMensagem.curvarDir:
1692                  System.out.println("yo");
1693                    buffer.putInt(msg.getArg1());
1694                    buffer.putInt(msg.getArg2());
1695                    break;
1696
1697                case iMensagem.curvarEsq:
1698                    buffer.putInt(msg.getArg1());
1699                    buffer.putInt(msg.getArg2());
1700                    break;
1701
1702                case iMensagem.vazia:
1703                    buffer.putInt(0);
1704                    System.out.println("limpei");
1705                    //id = -1;
1706                    break;
1707            }
1708
1709            putBuffer =++ putBuffer % nMensagens;
1710            msg.setId(id + 1);
1711
1712        }catch(Exception e) {
1713            e.printStackTrace();
1714        }
1715
1716    }
1717
1718    // fecha o canal entre o buffer e o ficheiro
1719    void fecharCanal() {
1720        if (canal != null)
1721            try {
1722                canal.close();
1723            } catch (IOException e) {
1724                canal = null;
1725            }
1726    }
1727
1728    /**Colocar todos os slots com mensagens vazias*/
1729    //Mts duvidas, gestao do ID, so tipo e id
1730    void inicializarBuffer() {
1731
1732        //1 msg - 4 * 4 = 16 bytes
1733        MensagemVazia msg = new MensagemVazia(0, iMensagem.vazia);
1734        System.out.print(BUFFER_MAX);
1735
1736        for(int i = 0; i < nMensagens; i++) {
1737            enviarMensagem(msg, false);
1738        }
```

```
1739        buffer.clear();
1740        putBuffer=0;
1741        getBuffer=0;
1742
1743    }
1744
1745    void limparLida(){
1746      System.out.println(getBuffer);
1747      if(getBuffer != 0)
1748        buffer.position((getBuffer-1) * 16);
1749      else
1750        buffer.position((7) * 16);
1751        buffer.putInt(getBuffer);
1752        buffer.putInt(iMensagem.vazia);
1753
1754    }
1755
1756
1757    /*public static void main(String[] args) {
1758
1759
1760      MensagemParar msgParar = new MensagemParar(0, 0, false);   // int id, int tipo, boolean sincrono
1761        MensagemReta msg     = new MensagemReta(1,1,20);         // int id, int tipo, int dist
1762        MensagemCurvar msgCD   = new MensagemCurvar(2, 2, 15, 15); //int id, int tipo, int raio, int ang
1763        MensagemCurvar msgCE   = new MensagemCurvar(3, 3, 12, 30);
1764        MensagemVazia msgV = new MensagemVazia(4,4);
1765
1766
1767        CanalComunicacao cc = new CanalComunicacao(8);
1768        cc.abrirCanalEscritor("comunicacao.dat");
1769
1770        cc.enviarMensagem(msg);
1771        Mensagem msg1 =cc.receberMensagemLeitor();
1772        System.out.println(msg1);
1773        cc.enviarMensagem(msgV);
1774        System.out.println(cc.receberMensagemLeitor());
1775        cc.enviarMensagem(msgCD);
1776        System.out.println(cc.receberMensagemLeitor());
1777        cc.enviarMensagem(msgCE);
1778        System.out.println(cc.receberMensagemLeitor());
1779
1780        cc.fecharCanal();
1781        }*/
1782 }
1783
1784 Classe iMensagem
1785
1786 package ptrabalho;
1787
1788 public interface iMensagem {
1789
1790     int parar          = 0;
1791     int reta           = 1;
1792     int curvarDir        = 2;
1793     int curvarEsq        = 3;
1794     int vazia             = 4;
1795     int pedir            = 5;
```

```
1796     int sensor            = 6;
1797     int workflow         = 7;
1798     int endWorkflow     = 8;
1799 }
1800
1801 Classe Mensagem
1802
1803 package ptrabalho;
1804
1805 public class Mensagem implements iMensagem{
1806     int tipo, id, arg1, arg2;
1807
1808     public Mensagem(int id, int tipo, int arg1, int arg2) {
1809         this.tipo = tipo;
1810         this.id = id;
1811         this.arg1 = arg1;
1812         this.arg2 = arg2;
1813     }
1814
1815     public Mensagem() {
1816
1817     }
1818
1819
1820     @Override
1821     public String toString() {
1822         return "Mensagem{" +
1823                 " id= " + id +
1824                 " tipo=" + tipo +
1825                 " arg1=" + arg1 +
1826                 " arg2=" + arg2 +
1827                 '}';
1828     }
1829
1830     public int getTipo() {
1831         return tipo;
1832     }
1833
1834     public void setTipo(int tipo) {
1835         this.tipo = tipo;
1836     }
1837
1838     public int getId() {
1839         return id;
1840     }
1841
1842     public void setId(int id) {
1843         this.id = id;
1844     }
1845
1846     public int getArg1() {
1847         return arg1;
1848     }
1849
1850     public void setArg1(int arg) {
1851         this.arg1 = arg;
1852     }
1853
1854     public int getArg2() {
1855         return arg2;
```

```
1856        }
1857
1858        public void setArg2(int arg) {
1859            this.arg2 = id;
1860        }
1861
1862        public boolean equals(Mensagem mensagem) {
1863
1864            if(mensagem==null) return false;
1865
1866            return (this.getId() == mensagem.getId());
1867        }
1868    }
1869
1870    Classe MensagemCurvar
1871
1872    package ptrabalho;
1873
1874    public class MensagemCurvar extends Mensagem{
1875      int raio, ang;
1876
1877      public MensagemCurvar(int id, int tipo, int raio, int ang) {
1878        super(id, tipo, raio, ang);
1879        this.raio = raio;
1880        this.ang  = ang;
1881      }
1882
1883      public int getRaio() {
1884        return raio;
1885      }
1886
1887      public void setRaio(int raio) {
1888        this.raio = raio;
1889      }
1890
1891      public int getAng() {
1892        return ang;
1893      }
1894
1895      public void setAng(int ang) {
1896        this.ang = ang;
1897      }
1898    }
1899
1900    Classe MensagemParar
1901
1902    package ptrabalho;
1903
1904    public class MensagemParar extends Mensagem{
1905
1906      boolean sincrono;
1907
1908      public MensagemParar(int id, int tipo, boolean sincrono) {
1909        super(id, tipo, 0, 0);
1910        this.sincrono = sincrono;
1911      }
1912
1913      @Override
1914      public String toString() {
1915        return super.toString() +" " +
```

```
1916          "sincrono=" + sincrono +
1917          '}';
1918    }
1919
1920    public boolean isSincrono() {
1921      return sincrono;
1922    }
1923
1924    public void setSincrono(boolean sincrono) {
1925      this.sincrono = sincrono;
1926    }
1927
1928  }
1929
1930  Classe MensagemReta
1931
1932  package ptrabalho;
1933
1934  public class MensagemReta extends Mensagem{
1935    int dist;
1936
1937    public MensagemReta(int id, int tipo, int dist) {
1938      super(id, tipo, dist, 0);
1939      this.dist = dist;
1940    }
1941
1942    public MensagemReta(int id, int tipo, int dist, int arg2) {
1943      super(id, tipo, dist, 0);
1944      this.dist = dist;
1945    }
1946
1947    public int getDist() {
1948      return dist;
1949    }
1950
1951    public void setDist(int dist) {
1952      this.dist = dist;
1953    }
1954
1955    @Override
1956    public String toString() {
1957      return super.toString() + " distancia= " + dist + "}";
1958    }
1959  }
1960
1961  Classe MensagemVazia
1962
1963  package ptrabalho;
1964
1965  public class MensagemVazia extends Mensagem{
1966      public MensagemVazia(int id, int tipo) {
1967          super(id, tipo, 0, 0);
1968      }
1969  }
```