



Visão do elevador (no CD)

I.1 Introdução

Este apêndice contém a implementação da classe **ElevatorView** (Fig. I.1). A familiaridade com as seções “Pensando em Objetos” presentes em todos os capítulos necessita da ampla compreensão do material contido neste apêndice. A classe **ElevatorView** é a maior na simulação. Para facilitar a discussão, dividimos a discussão da **ElevatorView** em cinco tópicos – *Objetos da classe*, *Constantes da classe*, *Construtor da classe*, *Tratamento de eventos* e *Diagrama de componentes revisitado*.

```
1 // ElevatorView.java
2 // Visão para ElevatorSimulation
3 package com.deitel.jhtp4.elevator.view;
4
5 // Pacotes do núcleo de Java
6 import java.awt.*;
7 import java.awt.event.*;
8 import java.util.*;
9 import java.applet.*;
10
11 // Pacote de extensão de Java
12 import javax.swing.*;
13
14 // Pacotes Deitel
15 import com.deitel.jhtp4.elevator.event.*;
16 import com.deitel.jhtp4.elevator.ElevatorConstants;
17
18 public class ElevatorView extends JPanel
19     implements ActionListener, ElevatorModelListener,
20         ElevatorConstants {
21
22     // dimensões da ElevatorView
23     private static final int VIEW_WIDTH = 800;
24     private static final int VIEW_HEIGHT = 435;
25
26     // deslocamento para posicionar Panels na ElevatorView
27     private static final int OFFSET = 10;
28 }
```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 1 de 16).

```

29 // Elevator pinta novamente os componentes a cada 50 ms
30 private static final int ANIMATION_DELAY = 50;
31
32 // constantes de distância horizontal
33 private static final int PERSON_TO_BUTTON_DISTANCE = 400;
34 private static final int BUTTON_TO_ELEVATOR_DISTANCE = 50;
35 private static final int PERSON_TO_ELEVATOR_DISTANCE =
36     PERSON_TO_BUTTON_DISTANCE + BUTTON_TO_ELEVATOR_DISTANCE;
37
38 // tempo de caminhada até o Button do andar e o Elevator
39 private static final int TIME_TO_BUTTON = 3000; // três segundos
40 private static final int TIME_TO_ELEVATOR = 1000; // um segundo
41
42 // tempo andando no Elevator (cinco segundos)
43 private static final int ELEVATOR_TRAVEL_TIME = 5000;
44
45 // imagens de Door para animação
46 private static final String doorFrames[] = {
47     "images/door1.png", "images/door2.png", "images/door3.png",
48     "images/door4.png", "images/door5.png" };
49
50 // imagens de Person para animação
51 private static final String personFrames[] = {
52     "images/bug1.png", "images/bug2.png", "images/bug3.png",
53     "images/bug4.png", "images/bug5.png", "images/bug6.png",
54     "images/bug7.png", "images/bug8.png" };
55
56 // imagens de Light para animação
57 private static final String lightFrames[] = {
58     "images/lightOff.png", "images/lightOn.png" };
59
60 // imagens da Light do Floor para animação
61 private static final String firstFloorLightFrames[] = {
62     "images/firstFloorLightOff.png",
63     "images/firstFloorLightOn.png" };
64
65 private static final String secondFloorLightFrames[] = {
66     "images/secondFloorLightOff.png",
67     "images/secondFloorLightOn.png", };
68
69 // imagens do Button do Floor para animação
70 private static final String floorButtonFrames[] = {
71     "images/floorButtonUnpressed.png",
72     "images/floorButtonPressed.png",
73     "images/floorButtonLit.png" };
74
75 // imagens do Button do Elevator para animação
76 private static final String elevatorButtonFrames[] = {
77     "images/elevatorButtonUnpressed.png",
78     "images/elevatorButtonPressed.png",
79     "images/elevatorButtonLit.png" };
80
81 // imagens da Bell para animação
82 private static final String bellFrames[] = {
83     "images/bell1.png", "images/bell2.png",
84     "images/bell3.png" };
85
86 private static final String floorImage =
87     "images/floor.png";
88 private static final String ceilingImage =

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 2 de 16).

```

89     "images/ceiling.png";
90     private static final String elevatorImage =
91         "images/elevator.png";
92     private static final String wallImage =
93         "images/wall.jpg";
94     private static final String elevatorShaftImage =
95         "images/elevatorShaft.png";
96
97     // arquivos de áudio
98     private static final String bellSound = "bell.wav";
99     private static final String doorOpenSound = "doorOpen.wav";
100    private static final String doorCloseSound = "doorClose.wav";
101    private static final String elevatorSound = "elevator.au";
102    private static final String buttonSound = "button.wav";
103    private static final String walkingSound = "walk.wav";
104
105    private static final String midiFile = "sounds/liszt.mid";
106
107    // ImagePanels para Floors, ElevatorShaft, parede e teto
108    private ImagePanel firstFloorPanel;
109    private ImagePanel secondFloorPanel;
110    private ImagePanel elevatorShaftPanel;
111    private ImagePanel wallPanel;
112    private ImagePanel ceilingPanel;
113
114    // MovingPanels para Elevator
115    private MovingPanel elevatorPanel;
116
117    // AnimatedPanels para Buttons, Bell, Lights e Door
118    private AnimatedPanel firstFloorButtonPanel;
119    private AnimatedPanel secondFloorButtonPanel;
120    private AnimatedPanel elevatorButtonPanel;
121    private AnimatedPanel bellPanel;
122    private AnimatedPanel elevatorLightPanel;
123    private AnimatedPanel firstFloorLightPanel;
124    private AnimatedPanel secondFloorLightPanel;
125    private AnimatedPanel doorPanel;
126
127    // List contendo AnimatedPanels para todos os objetos Person
128    private java.util.List personAnimatedPanels;
129
130    // AudioClips para efeitos de som
131    private AudioClip bellClip;
132    private AudioClip doorOpenClip;
133    private AudioClip doorCloseClip;
134    private AudioClip elevatorClip;
135    private AudioClip buttonClip;
136    private AudioClip walkClip;
137
138    // ElevatorMusic para tocar no Elevator
139    private ElevatorMusic elevatorMusic;
140
141    // Timer para o controlador da animação
142    private javax.swing.Timer animationTimer;
143
144    // distância do topo da tela para exibir Floors
145    private int firstFloorPosition;
146    private int secondFloorPosition;
147
148    // velocidade do Elevator

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 3 de 16).

```

149     private double elevatorVelocity;
150
151     // construtor ElevatorView
152     public ElevatorView()
153     {
154         // especifica leiaute null
155         super( null );
156
157         instantiatePanels();
158         placePanelsOnView();
159         initializeAudio();
160
161         // calcula a distância que o Elevator percorre
162         double floorDistance =
163             firstFloorPosition - secondFloorPosition;
164
165         // calcula o tempo necessário para o percurso
166         double time = ELEVATOR_TRAVEL_TIME / ANIMATION_DELAY;
167
168         // determina a velocidade do Elevator (taxa = distância / tempo)
169         elevatorVelocity = ( floorDistance + OFFSET ) / time;
170
171         // dispara a Thread de animação
172         startAnimation();
173
174     } // fim do construtor ElevatorView
175
176     // instancia todos os Panels (Floors, Elevator, etc.)
177     private void instantiatePanels()
178     {
179         // instancia ImagePanels representando os Floors
180         firstFloorPanel = new ImagePanel( 0, floorImage );
181         secondFloorPanel = new ImagePanel( 0, floorImage );
182
183         // calcula as posições do primeiro e do segundo Floor
184         firstFloorPosition =
185             VIEW_HEIGHT - firstFloorPanel.getHeight();
186         secondFloorPosition =
187             ( int ) ( firstFloorPosition / 2 ) - OFFSET;
188
189         firstFloorPanel.setPosition( 0, firstFloorPosition );
190         secondFloorPanel.setPosition( 0, secondFloorPosition );
191
192         wallPanel = new ImagePanel( 0, wallImage );
193
194         // cria e posiciona ImagePanel para ElevatorShaft
195         elevatorShaftPanel =
196             new ImagePanel( 0, elevatorShaftImage );
197
198         double xPosition = PERSON_TO_ELEVATOR_DISTANCE + OFFSET;
199         double yPosition =
200             firstFloorPosition - elevatorShaftPanel.getHeight();
201
202         elevatorShaftPanel.setPosition( xPosition, yPosition );
203
204         // cria e posiciona ImagePanel para o teto
205         ceilingPanel = new ImagePanel( 0, ceilingImage );
206
207         yPosition = elevatorShaftPanel.getPosition().getY() -
208             ceilingPanel.getHeight();

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 4 de 16).

```

209
210     ceilingPanel.setPosition( xPositon, yPositon );
211
212     // cria e posiciona MovingPanel para Elevator
213     elevatorPanel = new MovingPanel( 0, elevatorImage );
214
215     yPositon = firstFloorPosition - elevatorPanel.getHeight();
216
217     elevatorPanel.setPosition( xPositon, yPositon );
218
219     // cria e posiciona o Button do primeiro Floor
220     firstFloorButtonPanel =
221         new AnimatedPanel( 0, floorButtonFrames );
222
223     xPositon = PERSON_TO_BUTTON_DISTANCE + 2 * OFFSET;
224     yPositon = firstFloorPosition - 5 * OFFSET;
225     firstFloorButtonPanel.setPosition( xPositon, yPositon );
226
227     int floorButtonPressedFrameOrder[] = { 0, 1, 2 };
228     firstFloorButtonPanel.addFrameSequence(
229         floorButtonPressedFrameOrder );
230
231     // cria e posiciona o Button do segundo Floor
232     secondFloorButtonPanel =
233         new AnimatedPanel( 1, floorButtonFrames );
234
235     xPositon = PERSON_TO_BUTTON_DISTANCE + 2 * OFFSET;
236     yPositon = secondFloorPosition - 5 * OFFSET;
237     secondFloorButtonPanel.setPosition( xPositon, yPositon );
238
239     secondFloorButtonPanel.addFrameSequence(
240         floorButtonPressedFrameOrder );
241
242     // cria e posiciona as Lights dos Floors
243     firstFloorLightPanel =
244         new AnimatedPanel( 0, firstFloorLightFrames );
245
246     xPositon = elevatorPanel.getLocation().x - 4 * OFFSET;
247     yPositon =
248         firstFloorButtonPanel.getLocation().y - 10 * OFFSET;
249     firstFloorLightPanel.setPosition( xPositon, yPositon );
250
251     secondFloorLightPanel =
252         new AnimatedPanel( 1, secondFloorLightFrames );
253
254     yPositon =
255         secondFloorButtonPanel.getLocation().y - 10 * OFFSET;
256     secondFloorLightPanel.setPosition( xPositon, yPositon );
257
258     // cria e posiciona os AnimatedPanels de Doors
259     doorPanel = new AnimatedPanel( 0, doorFrames );
260     int doorOpenedFrameOrder[] = { 0, 1, 2, 3, 4 };
261     int doorClosedFrameOrder[] = { 4, 3, 2, 1, 0 };
262     doorPanel.addFrameSequence( doorOpenedFrameOrder );
263     doorPanel.addFrameSequence( doorClosedFrameOrder );
264
265     // determina onde a Door está localizada em relação ao Elevator
266     yPositon =
267         elevatorPanel.getHeight() - doorPanel.getHeight();
268

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 5 de 16).

```

269     doorPanel.setPosition( 0, yPosition );
270
271     // cria e posiciona os AnimatedPanels de Lights
272     elevatorLightPanel = new AnimatedPanel( 0, lightFrames );
273     elevatorLightPanel.setPosition( OFFSET, 5 * OFFSET );
274
275     // cria e posiciona o AnimatedPanel de Bell
276     bellPanel = new AnimatedPanel( 0, bellFrames );
277
278     yPosition = elevatorLightPanel.getPosition().getY() +
279         elevatorLightPanel.getHeight() + OFFSET;
280
281     bellPanel.setPosition( OFFSET, yPosition );
282     int bellRingAnimation[] = { 0, 1, 0, 2 };
283     bellPanel.addFrameSequence( bellRingAnimation );
284
285     // cria e posiciona o AnimatedPanel do Button do Elevator
286     elevatorButtonPanel =
287         new AnimatedPanel( 0, elevatorButtonFrames );
288
289     yPosition = elevatorPanel.getHeight() - 6 * OFFSET;
290     elevatorButtonPanel.setPosition( 10 * OFFSET, yPosition );
291
292     int buttonPressedFrameOrder[] = { 0, 1, 2 };
293     elevatorButtonPanel.addFrameSequence(
294         buttonPressedFrameOrder );
295
296     // cria List para armazenar os AnimatedPanels de Person
297     personAnimatedPanels = new ArrayList();
298
299 } // fim do método instantiatePanels
300
301 // coloca todos os Panels na ElevatorView
302 private void placePanelsOnView()
303 {
304     // adiciona Panels à ElevatorView
305     add( firstFloorPanel );
306     add( secondFloorPanel );
307     add( ceilingPanel );
308     add( elevatorPanel );
309     add( firstFloorButtonPanel );
310     add( secondFloorButtonPanel );
311     add( firstFloorLightPanel );
312     add( secondFloorLightPanel );
313     add( elevatorShaftPanel );
314     add( wallPanel );
315
316     // adiciona Panels ao MovingPanel de Elevator
317     elevatorPanel.add( doorPanel );
318     elevatorPanel.add( elevatorLightPanel );
319     elevatorPanel.add( bellPanel );
320     elevatorPanel.add( elevatorButtonPanel );
321
322 } // fim do método placePanelsOnView
323
324 // obtém efeitos de som e elevatorMusic
325 private void initializeAudio()
326 {
327     // cria efeitos de som AudioClip a partir de arquivos de áudio
328     SoundEffects sounds = new SoundEffects();

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 6 de 16).

```

329     sounds.setPathPrefix( "sounds/" );
330
331     bellClip = sounds.getAudioClip( bellSound );
332     doorOpenClip = sounds.getAudioClip( doorOpenSound );
333     doorCloseClip = sounds.getAudioClip( doorCloseSound );
334     elevatorClip = sounds.getAudioClip( elevatorSound );
335     buttonClip = sounds.getAudioClip( buttonSound );
336     walkClip = sounds.getAudioClip( walkingSound );
337
338     // cria reproduzidor de MIDI usando Java Media Framework
339     elevatorMusic = new ElevatorMusic( midiFile );
340     elevatorMusic.open();
341
342 } // fim do método initializeAudio
343
344 // inicia a animação desenhando imagens repetidamente na tela
345 public void startAnimation()
346 {
347     if ( animationTimer == null ) {
348         animationTimer =
349             new javax.swing.Timer( ANIMATION_DELAY, this );
350         animationTimer.start();
351     }
352     else
353         if ( !animationTimer.isRunning() )
354             animationTimer.restart();
355 }
356
357 // faz parar a animação
358 public void stopAnimation()
359 {
360     animationTimer.stop();
361 }
362
363 // atualiza a animação de AnimatedPanels em resposta ao Timer
364 public void actionPerformed( ActionEvent actionEvent )
365 {
366     elevatorPanel.animate();
367
368     firstFloorButtonPanel.animate();
369     secondFloorButtonPanel.animate();
370
371     Iterator iterator = getPersonAnimatedPanelsIterator();
372
373     while ( iterator.hasNext() ) {
374         // obtém o AnimatedPanel da Person do Set
375         AnimatedPanel personPanel =
376             ( AnimatedPanel ) iterator.next();
377
378         personPanel.animate(); // atualiza painel
379     }
380
381     repaint(); // pinta todos os componentes
382 } // fim do método actionPerformed
383
384 private Iterator getPersonAnimatedPanelsIterator()
385 {

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 7 de 16).

```

389         // obtém iterador de List
390         synchronized( personAnimatedPanels )
391         {
392             return new ArrayList( personAnimatedPanels ).iterator();
393         }
394     }
395
396     // faz parar clipe de som de Person caminhando
397     private void stopWalkingSound()
398     {
399         // faz parar reprodução do som de passos
400         walkClip.stop();
401
402         Iterator iterator = getPersonAnimatedPanelsIterator();
403
404         // mas se a Person ainda estiver caminhando, continuando reproduzindo
405         while ( iterator.hasNext() ) {
406             AnimatedPanel panel = ( AnimatedPanel ) iterator.next();
407
408             if ( panel.getXVelocity() != 0 )
409                 walkClip.loop();
410         }
411     } // fim do método stopWalkingSound
412
413     // devolve o AnimatedPanel da Person com o identificador apropriado
414     private AnimatedPanel getPersonPanel( PersonMoveEvent event )
415     {
416         Iterator iterator = getPersonAnimatedPanelsIterator();
417
418         while ( iterator.hasNext() ) {
419
420             // obtém o próximo AnimatedPanel
421             AnimatedPanel personPanel =
422                 ( AnimatedPanel ) iterator.next();
423
424             // devolve AnimatedPanel com identificador que coincide
425             if ( personPanel.getID() == event.getID() )
426                 return personPanel;
427         }
428
429         // devolve null se nenhum coincidir com o identificador correto
430         return null;
431     } // fim do método getPersonPanel
432
433     // invocado quando o Elevator partiu do Floor
434     public void elevatorDeparted( ElevatorMoveEvent moveEvent )
435     {
436
437         String location =
438             moveEvent.getLocation().getLocationName();
439
440         // determina se a Person está no Elevator
441         Iterator iterator = getPersonAnimatedPanelsIterator();
442
443         while ( iterator.hasNext() ) {
444
445             AnimatedPanel personPanel =
446                 ( AnimatedPanel ) iterator.next();
447
448             double yPosition = personPanel.getPosition().getY();

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 8 de 16).

```

449         String panelLocation;
450
451         // determina em que Floor a Person entrou
452         if ( yPosition > secondFloorPosition )
453             panelLocation = FIRST_FLOOR_NAME;
454         else
455             panelLocation = SECOND_FLOOR_NAME;
456
457         int xPosition =
458             ( int ) personPanel.getPosition().getX();
459
460         // se Person estiver dentro do Elevator
461         if ( panelLocation.equals( location )
462             && xPosition > PERSON_TO_BUTTON_DISTANCE + OFFSET ) {
463
464             // remove o AnimatedPanel de Person da ElevatorView
465             remove( personPanel );
466
467             // adiciona o AnimatedPanel da Person ao Elevator
468             elevatorPanel.add( personPanel, 1 );
469             personPanel.setLocation( 2 * OFFSET, 9 * OFFSET );
470             personPanel.setMoving( false );
471             personPanel.setAnimating( false );
472             personPanel.setVelocity( 0, 0 );
473             personPanel.setCurrentFrame( 1 );
474         }
475     } // fim do laço while
476
477     // determina a velocidade do Elevator dependendo do Floor
478     if ( location.equals( FIRST_FLOOR_NAME ) )
479         elevatorPanel.setVelocity( 0, -elevatorVelocity );
480     else
481
482         if ( location.equals( SECOND_FLOOR_NAME ) )
483             elevatorPanel.setVelocity( 0, elevatorVelocity );
484
485     // começa a movimentar o Elevator e reproduzir elevatorMusic
486     elevatorPanel.setMoving( true );
487
488     if ( elevatorClip != null )
489         elevatorClip.play();
490
491     elevatorMusic.play();
492 } // fim do método elevatorDeparted
493
494 // invocado quando o Elevator chegou ao Floor de destino
495 public void elevatorArrived( ElevatorMoveEvent moveEvent )
496 {
497     // faz parar Elevator e a música
498     elevatorPanel.setMoving( false );
499     elevatorMusic.getSequencer().stop();
500
501     double xPosition = elevatorPanel.getPosition().getX();
502     double yPosition;
503
504     // ajusta a posição do Elevator para o primeiro ou o segundo Floor
505     if ( elevatorPanel.getYVelocity() < 0 )
506         yPosition =
507             secondFloorPosition - elevatorPanel.getHeight();
508

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 9 de 16).

```

509         else
510             yPosition =
511                 firstFloorPosition - elevatorPanel.getHeight();
512
513         elevatorPanel.setPosition( xPosition, yPosition );
514
515     } // fim do método elevatorArrived
516
517     // invocado quando Person foi criada no modelo
518     public void personCreated( PersonMoveEvent personEvent )
519     {
520         int personID = personEvent.getID();
521
522         String floorLocation =
523             personEvent.getLocation().getLocationName();
524
525         // cria AnimatedPanel representando a Person
526         AnimatedPanel personPanel =
527             new AnimatedPanel( personID, personFrames );
528
529         // determina se a Person deve ser desenhada inicialmente
530         // xPosition negativa assegura que Person é desenhada fora da tela
531         double xPosition = - personPanel.getWidth();
532         double yPosition = 0;
533
534         if ( floorLocation.equals( FIRST_FLOOR_NAME ) )
535             yPosition = firstFloorPosition +
536                 ( firstFloorPanel.getHeight() / 2 );
537         else
538
539             if ( floorLocation.equals( SECOND_FLOOR_NAME ) )
540                 yPosition = secondFloorPosition +
541                     ( secondFloorPanel.getHeight() / 2 );
542
543         yPosition -= personPanel.getHeight();
544
545         personPanel.setPosition( xPosition, yPosition );
546
547         // adiciona algumas animações para cada Person
548         int walkFrameOrder[] = { 1, 0, 1, 2 };
549         int pressButtonFrameOrder[] = { 1, 3, 3, 4, 4, 1 };
550         int walkAwayFrameOrder[] = { 6, 5, 6, 7 };
551         personPanel.addFrameSequence( walkFrameOrder );
552         personPanel.addFrameSequence( pressButtonFrameOrder );
553         personPanel.addFrameSequence( walkAwayFrameOrder );
554
555         // faz a Person começar a caminhar para o Elevator
556         personPanel.playAnimation( 0 );
557         personPanel.setLoop( true );
558         personPanel.setAnimating( true );
559         personPanel.setMoving( true );
560
561         // determina a velocidade da Person
562         double time =
563             ( double ) ( TIME_TO_BUTTON / ANIMATION_DELAY );
564
565         double xDistance = PERSON_TO_BUTTON_DISTANCE -
566             2 * OFFSET + personPanel.getSize().width;
567         double xVelocity = xDistance / time;
568

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 10 de 16).

```

569     personPanel.setVelocity( xVelocity, 0 );
570     personPanel.setAnimationRate( 1 );
571
572     walkClip.loop();    // reproduz clipe de som de Person caminhando
573
574     // armazena em personAnimatedPanels
575     synchronized( personAnimatedPanels )
576     {
577         personAnimatedPanels.add( personPanel );
578     }
579
580     add( personPanel, 0 );
581
582 } // fim do método personCreated
583
584 // invocado quando Person chegou no Elevator
585 public void personArrived( PersonMoveEvent personEvent )
586 {
587     // encontra Panel associado com a Person que disparou o evento
588     AnimatedPanel panel = getPersonPanel( personEvent );
589
590     if ( panel != null ) {    // se Person existir
591
592         // Person para junto ao Button do Floor
593         panel.setMoving( false );
594         panel.setAnimating( false );
595         panel.setCurrentFrame( 1 );
596         stopWalkingSound();
597
598         double xPosition = PERSON_TO_BUTTON_DISTANCE -
599             ( panel.getSize().width / 2 );
600         double yPosition = panel.getPosition().getY();
601
602         panel.setPosition( xPosition, yPosition );
603     }
604 } // fim do método personArrived
605
606 // invocado quando Person pressionou Button
607 public void personPressedButton( PersonMoveEvent personEvent )
608 {
609     // encontra Panel associado com a Person que disparou o evento
610     AnimatedPanel panel = getPersonPanel( personEvent );
611
612     if ( panel != null ) {    // se Person existir
613
614         // Person pára de caminhar e pressiona Button
615         panel.setLoop( false );
616         panel.playAnimation( 1 );
617
618         panel.setVelocity( 0, 0 );
619         panel.setMoving( false );
620         panel.setAnimating( true );
621         stopWalkingSound();
622     }
623 } // fim do método personPressedButton
624
625 // invocado quando Person começou a entrar no Elevator
626 public void personEntered( PersonMoveEvent personEvent )
627 {
628     // encontra Panel associado com a Person que disparou o evento

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 11 de 16).

```

629     AnimatedPanel panel = getPersonPanel( personEvent );
630
631     if ( panel != null ) {
632
633         // determina a velocidade
634         double time = TIME_TO_ELEVATOR / ANIMATION_DELAY;
635
636         double distance =
637             elevatorPanel.getPosition().getX() -
638             panel.getPosition().getX() + 2 * OFFSET;
639
640         panel.setVelocity( distance / time, -1.5 );
641
642         // Person começa a caminhar
643         panel.setMoving( true );
644         panel.playAnimation( 0 );
645         panel.setLoop( true );
646     }
647 } // fim do método personEntered
648
649 // invocado quando a Person saiu do Elevator
650 public void personDeparted( PersonMoveEvent personEvent)
651 {
652     // encontra Panel associado com a Person que disparou o evento
653     AnimatedPanel panel = getPersonPanel( personEvent );
654
655     if ( panel != null ) { // se Person existir
656
657         // determina velocidade (na direção oposta)
658         double time = TIME_TO_BUTTON / ANIMATION_DELAY;
659         double xVelocity = - PERSON_TO_BUTTON_DISTANCE / time;
660
661         panel.setVelocity( xVelocity, 0 );
662
663         // remove Person do Elevator
664         elevatorPanel.remove( panel );
665
666         double xPosition =
667             PERSON_TO_ELEVATOR_DISTANCE + 3 * OFFSET;
668         double yPosition = 0;
669
670         String floorLocation =
671             personEvent.getLocation().getLocationName();
672
673         // determina o Floor para o qual a Person sai
674         if ( floorLocation.equals( FIRST_FLOOR_NAME ) )
675             yPosition = firstFloorPosition +
676                 ( firstFloorPanel.getHeight() / 2 );
677         else
678
679             if ( floorLocation.equals( SECOND_FLOOR_NAME ) )
680                 yPosition = secondFloorPosition +
681                     ( secondFloorPanel.getHeight() / 2 );
682
683         yPosition -= panel.getHeight();
684
685         panel.setPosition( xPosition, yPosition );
686
687         // adiciona Person à ElevatorView
688         add( panel, 0 );

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 12 de 16).

```

689
690         // Person começa a caminhar
691         panel.setMoving( true );
692         panel.setAnimating( true );
693         panel.playAnimation( 2 );
694         panel.setLoop( true );
695         walkClip.loop();
696     }
697 } // fim do método PersonDeparted
698
699 // invocado quando a Person saiu da simulação
700 public void personExited( PersonMoveEvent personEvent)
701 {
702     // encontra Panel associado com a Person que disparou o moveEvent
703     AnimatedPanel panel = getPersonPanel( personEvent );
704
705     if ( panel != null ) { // se Person existir
706
707         panel.setMoving( false );
708         panel.setAnimating( false );
709
710         // remove Person permanentemente e faz parar som de passos
711         synchronized( personAnimatedPanels )
712         {
713             personAnimatedPanels.remove( panel );
714         }
715         remove( panel );
716         stopWalkingSound();
717     }
718 } // fim do método personExited
719
720 // invocado quando a Door abriu no modelo
721 public void doorOpened( DoorEvent doorEvent )
722 {
723     // obtém Location do DoorEvent
724     String location =
725         doorEvent.getLocation().getLocationName();
726
727     // reproduz animação da Door abrindo
728     doorPanel.playAnimation( 0 );
729     doorPanel.setAnimationRate( 2 );
730     doorPanel.setDisplayLastFrame( true );
731
732     // reproduz clipe de som da Door abrindo
733     if ( doorOpenClip != null )
734         doorOpenClip.play();
735
736 } // fim do método doorOpened
737
738 // invocado quando a Door fechou no modelo
739 public void doorClosed( DoorEvent doorEvent )
740 {
741     // obtém Location do DoorEvent
742     String location =
743         doorEvent.getLocation().getLocationName();
744
745     // reproduz animação da Door fechando
746     doorPanel.playAnimation( 1 );
747     doorPanel.setAnimationRate( 2 );
748     doorPanel.setDisplayLastFrame( true );

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 13 de 16).

```

749
750     // reproduz clipe de som da Door fechando
751     if ( doorCloseClip != null )
752         doorCloseClip.play();
753
754 } // fim do método doorClosed
755
756 // invocado quando o Button foi pressionado no modelo
757 public void buttonPressed( ButtonEvent buttonEvent )
758 {
759     // obtém Location do ButtonEvent
760     String location =
761         buttonEvent.getLocation().getLocationName();
762
763     // pressiona Button do Elevator se foi do Elevator
764     if ( location.equals( ELEVATOR_NAME ) ) {
765         elevatorButtonPanel.playAnimation( 0 );
766         elevatorButtonPanel.setDisplayLastFrame( true );
767     }
768
769     // pressiona Button do Floor se foi do Floor
770     else
771     {
772         if ( location.equals( FIRST_FLOOR_NAME ) ) {
773             firstFloorButtonPanel.playAnimation( 0 );
774             firstFloorButtonPanel.setDisplayLastFrame( true );
775         }
776         else
777         {
778             if ( location.equals( SECOND_FLOOR_NAME ) ) {
779                 secondFloorButtonPanel.playAnimation( 0 );
780                 secondFloorButtonPanel.setDisplayLastFrame( true );
781             }
782
783             if ( buttonClip != null )
784                 buttonClip.play(); // reproduz clipe de som de pressionamento do Button
785
786 } // fim do método buttonPressed
787
788 // invocado quando o Button foi desligado no modelo
789 public void buttonReset( ButtonEvent buttonEvent )
790 {
791     // obtém Location do ButtonEvent
792     String location =
793         buttonEvent.getLocation().getLocationName();
794
795     // desliga ElevatorButton se foi do Elevator
796     if ( location.equals( ELEVATOR_NAME ) ) {
797
798         // volta para a primeira frame se ainda estiver animando
799         if ( elevatorButtonPanel.isAnimating() )
800             elevatorButtonPanel.setDisplayLastFrame( false );
801         else
802             elevatorButtonPanel.setCurrentFrame( 0 );
803     }
804
805     // desliga FloorButton se foi de um Floor
806     else
807

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 14 de 16).

```

808         if ( location.equals( FIRST_FLOOR_NAME ) ) {
809             // retorna para o primeiro frame se ainda estiver animando
810             if ( firstFloorButtonPanel.isAnimating() )
811                 firstFloorButtonPanel.setDisplayLastFrame(
812                     false );
813             else
814                 firstFloorButtonPanel.setCurrentFrame( 0 );
815         }
816     else
817     {
818         if ( location.equals( SECOND_FLOOR_NAME ) ) {
819             // retorna para o primeiro frame se ainda estiver animando
820             if ( secondFloorButtonPanel.isAnimating() )
821                 secondFloorButtonPanel.setDisplayLastFrame(
822                     false );
823             else
824                 secondFloorButtonPanel.setCurrentFrame( 0 );
825         }
826     }
827 } // fim do método buttonReset
828
829 // invocado quando a Bell tocou no modelo
830 public void bellRang( BellEvent bellEvent )
831 {
832     bellPanel.playAnimation( 0 ); // anima a Bell
833
834     if ( bellClip != null ) // reproduz clipe de som da Bell
835         bellClip.play();
836 }
837
838 // invocado quando uma Light acendeu no modelo
839 public void lightTurnedOn( LightEvent lightEvent )
840 {
841     // acende a Light no Elevator
842     elevatorLightPanel.setCurrentFrame( 1 );
843
844     String location =
845         lightEvent.getLocation().getLocationName();
846
847     // acende a Light no primeiro ou segundo Floor
848     if ( location.equals( FIRST_FLOOR_NAME ) )
849         firstFloorLightPanel.setCurrentFrame( 1 );
850     else
851         if ( location.equals( SECOND_FLOOR_NAME ) )
852             secondFloorLightPanel.setCurrentFrame( 1 );
853 } // fim do método lightTurnedOn
854
855

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 15 de 16).

```

860 // invocado quando uma Light apagou no modelo
861 public void lightTurnedOff( LightEvent lightEvent )
862 {
863     // desliga Light no Elevator
864     elevatorLightPanel.setCurrentFrame( 0 );
865
866     String location =
867         lightEvent.getLocation().getLocationName();
868
869     // desliga Light no primeiro ou segundo Floor
870     if ( location.equals( FIRST_FLOOR_NAME ) )
871         firstFloorLightPanel.setCurrentFrame( 0 );
872
873     else
874
875         if ( location.equals( SECOND_FLOOR_NAME ) )
876             secondFloorLightPanel.setCurrentFrame( 0 );
877
878 } // fim do método lightTurnedOff
879
880 // devolve tamanho preferencial da ElevatorView
881 public Dimension getPreferredSize()
882 {
883     return new Dimension( VIEW_WIDTH, VIEW_HEIGHT );
884 }
885
886 // devolve tamanho mínimo da ElevatorView
887 public Dimension getMinimumSize()
888 {
889     return getPreferredSize();
890 }
891
892 // devolve tamanho máximo da ElevatorView
893 public Dimension getMaximumSize()
894 {
895     return getPreferredSize();
896 }
897 }

```

Fig. I.1 ElevatorView exibe o modelo de simulação do elevador (parte 16 de 16).

I.2 Objetos da classe

A **ElevatorView** é um **JPanel** com uma série de outros **JPanel** “filhos” a ele adicionados. Cada **JPanel** fornece uma representação visual de um objeto do modelo. Por exemplo, a **ElevatorView** contém **ImagePanels**, **MovingPanels** e **AnimatedPanels** para representar o **Elevator**, as **Persons**, o **ElevatorShaft**, os **Buttons** nos **Floors**, o **Button** no **Elevator**, as **Doors** nos **Floors**, a **Door** no **Elevator**, as **Lights** nos **Floors**, os dois **Floors** e a **Bell**. A Fig. I.2 lista os objetos de **ElevatorView** e seus correspondentes no modelo.

O objeto (no modelo) da Classe ...	é representado pelo objeto (na visão) ...	da Classe ...
Floor	firstFloorPanel secondFloorPanel	ImagePanel
ElevatorShaft	elevatorShaftPanel	ImagePanel
Elevator	elevatorPanel	MovingPanel
Button (no Floor)	firstFloorButtonPanel secondFloorButtonPanel	AnimatedPanel
Button (no Elevator)	elevatorButtonPanel	AnimatedPanel
Bell	bellPanel	AnimatedPanel
Light	firstFloorLightPanel secondFloorLightPanel	AnimatedPanel
Door (no Elevator)	doorPanel	AnimatedPanel
Door (no Floor)	<não representada>	< não representada>
Person	personAnimatedPanels	List (de AnimatedPanels)

Fig. I.2 Objetos na `ElevatorView` que representam objetos no modelo.

As linhas 108 a 128 da classe `ElevatorView` declaram os objetos na segunda coluna da Fig. I.2. O `firstFloorPanel` (linha 108), `secondFloorPanel` (linha 109) e o `elevatorShaftPanel` (linha 110) são `ImagePanels`, porque nem os `Floors` nem o `ElevatorShaft` se movem na simulação. O `elevatorPanel` (linha 115) é um `MovingPanel`, porque a única função do `Elevator` é se mover entre `Floors`. O `firstFloorButtonPanel` (linha 118), o `secondFloorButtonPanel` (linha 119) e o `elevatorButtonPanel` (linha 120) são `AnimatedPanels`, porque cada objeto é animado quando o `Button` associado no modelo é pressionado ou desligado. O `bellPanel` (linha 121) é um `AnimatedPanel` para animar o toque da `Bell`. O `firstFloorLightPanel` (linha 123) e o `secondFloorLightPanel` (linha 124) são `AnimatedPanels` porque estes objetos são animados quando a `Light` associada acende ou apaga. O `doorPanel` (linha 125) é um `AnimatedPanel` para animar a abertura e o fechamento da `Door`. Observe que a `ElevatorView` mostra somente a `Door` no `Elevator`. A `ElevatorView` não mostra as `Doors` nos `Floors`, o que nos permite mostrar o interior do `Elevator` (estas `Doors` obstruiriam a visão dos objetos dentro do `Elevator`). Finalmente, a `personAnimatedPanels` (linha 128) é uma `List` de `AnimatedPanels`, porque podem existir diversos objetos `Person` durante a execução – a `ElevatorView` pode precisar armazenar dinamicamente os `AnimatedPanels` associados com as `Persons` no modelo.

O objeto (no modelo) da Classe ...	é representado pelo objeto (na visão) ...	da Classe ...
<não representada>	elevatorLightPanel	AnimatedPanel
<não representada>	ceilingPanel	ImagePanel
<não representada>	wallPanel	ImagePanel

Fig. I.3 Objetos na `ElevatorView` não representados no modelo.

Adicionamos à **ElevatorView** mais três elementos que supomos serem parte do **Elevator** (Fig. I.3), embora o modelo não represente estes elementos – uma luz dentro do **Elevator** do tipo **AnimatedPanel**, chamada **elevatorLightPanel** (linha 122), um teto sobre o **Elevator**, do tipo **ImagePanel**, chamado **ceilingPanel** (linha 112) e papel de parede dentro do edifício, do tipo **ImagePanel**, chamado **wallPanel** (linha 111).

Além disso, o diagrama de classes da Fig. 22.9 mostra que a **ElevatorView** contém uma instância de cada uma das classes **SoundEffects** e **ElevatorMusic**. O objeto **SoundEffects** gera os **AudioClips** usados para reproduzir efeitos de som, como a **Door** abrindo e uma **Person** caminhando. As linhas 131 a 136 declaram todos os **AudioClips**, a linha 139 declara o objeto **ElevatorMusic** e a linha 328 (no método **initializeAudio**, que discutiremos mais tarde nesta seção) declara o objeto **SoundEffects**.

1.3 Constantes da classe

A **ElevatorView** usa constantes para especificar ou obter informações como

- A posição inicial de objetos na **ElevatorView**;
- A frequência com a qual **ElevatorView** redesenha a tela (taxa de animação);
- Os nomes dos arquivos de imagem usados pelos **ImagePanels**;
- Os nomes dos arquivos de som usados pelos objetos **SoundEffects** e pelo **ElevatorMusic**;
- As distâncias, em *pixels*, que os **ImagePanels** que representam o **Elevator** e **Person** devem percorrer;
- Os tempos necessários para percorrer estas distâncias.

As linhas 23 e 24 declaram as constantes **int VIEW_WIDTH** e **VIEW_HEIGHT**, que especificam as dimensões da **ElevatorView**. O método **getPreferredSize** (linhas 881 a 884) devolve esta dimensão. O método **pack** da classe **ElevatorSimulation** usa este método para obter as dimensões da **ElevatorView** para posicionar a **ElevatorView** na GUI apropriadamente.

A **ElevatorView** tem um leiaute **null**, de modo que pode posicionar **ImagePanels** em qualquer coordenada *x-y* na **ElevatorView**. A linha 27 da classe **ElevatorView** declara a constante **int OFFSET**, que ajuda a determinar as posições exatas dos objetos na **ElevatorView**. A linha 30 declara a constante **int ANIMATION_DELAY**, que especifica o número de milissegundos entre as *frames* de animação. Em nossa simulação, inicializamos **ANIMATION_DELAY** com 50 milissegundos. As linhas 46 a 95 declaram constantes **String** que especificam os arquivos de imagem usados para instanciar os **ImagePanels**. As linhas 98 a 105 declaram as constantes **String** que especificam os arquivos de áudio usados para instanciar os **AudioClips** e a **ElevatorMusic**.

A linha 33 declara a constante **int PERSON_TO_BUTTON_DISTANCE**, que representa a distância horizontal entre a posição na tela do **firstFloorButtonPanel** ou do **secondFloorButtonPanel** e a posição inicial na tela de um **AnimatedPanel** associado com **Person**. Este **AnimatedPanel** utiliza esta constante para calcular a distância a percorrer até o **firstFloorButtonPanel** ou o **secondFloorButtonPanel**. O **firstFloorButtonPanel** e o **secondFloorButtonPanel** usam esta constante para posicionar a si mesmos na tela. A linha 34 declara a constante **int BUTTON_TO_ELEVATOR_DISTANCE**, que descreve a distância horizontal entre o **firstFloorButtonPanel** ou o **secondFloorButtonPanel** e o **elevatorPanel**. O **AnimatedPanel** associado com uma **Person** utiliza esta constante para determinar a distância a percorrer ao se entrar no **elevatorPanel**.

A linha 39 declara a constante **int TIME_TO_BUTTON**, que representa o tempo de percurso deste **AnimatedPanel** até o **firstFloorButtonPanel** ou o **secondFloorButtonPanel**. A linha 40 declara a constante **int TIME_TO_ELEVATOR**, que representa o tempo que o **AnimatedPanel** associado com uma **Person** precisa para entrar no **elevatorPanel** a partir do **firstFloorButtonPanel** ou o **secondFloorButtonPanel**. Usando a equação $\text{taxa} = \text{distância} / \text{tempo}$, o **AnimatedPanel** associado com a **Person** pode determinar a velocidade necessária para fazer o percurso. De maneira semelhante, a linha 43 declara a constante **int ELEVATOR_TRAVEL_TIME**, que representa o tempo de percurso do **elevatorPanel** entre o **firstFloorPanel** e o **secondFloorPanel** – as linhas 166 a 169 usam esta constante para determinar o atributo **double elevatorVelocity** (linha 149).

1.4 Construtor da classe

As responsabilidades do construtor de **ElevatorView** (linhas 152 a 174) são

- Instanciar todos os **ImagePanels**;
- Adicionar todos os **ImagePanels** à **ElevatorView**;
- Inicializar os objetos de áudio;
- Calcular a velocidade inicial e a distância a percorrer do **elevatorPanel**;
- Dar partida ao **Timer** da animação.

A linha 157 chama o método **private instantiatePanels** (linhas 177 a 299), o qual instancia todos os **ImagePanels** na **ElevatorView**. As linhas 180 e 181 instanciam o **firstFloorPanel** e o **secondFloorPanel**, e as linhas 184 a 190 ajustam as posições destes objetos – a **ElevatorView** posiciona o **firstFloorPanel** na parte inferior da tela e posiciona o **secondFloorPanel** no centro vertical da tela. A linha 192 instancia o **ImagePanel wallPanel**. A **ElevatorView** não precisa calcular a posição para o **wallPanel**, porque a posição *default* na tela para o **wallPanel** (i. e., **xPosition** = 0, **yPosition** = 0) está correta. As linhas 195 a 202 e as linhas 205 a 210 instanciam e posicionam os **ImagePanels elevatorShaftPanel** e **ceilingPanel**, respectivamente. A **ElevatorView** posiciona o **elevatorShaftPanel** à direita na tela e posiciona o **ceilingPanel** acima do **elevatorShaftPanel**. As linhas 213 a 217 instanciam o **elevatorPanel** e o posicionam sobre o **elevatorShaftPanel**, acima do **firstFloorPanel**. As linhas 220 a 229 instanciam o **AnimatedPanel firstFloorButtonPanel**, colocam-no em seguida no **elevatorShaftPanel** e depois criam uma sequência de *frames* para a animação de **Button** pressionado. As linhas 232 a 240 executam as mesmas ações sobre o **secondFloorButtonPanel**. As linhas 243 a 249 instanciam o **AnimatedPanel firstFloorLightPanel** e o colocam à esquerda do **elevatorShaftPanel**, mas acima do **firstFloorButtonPanel**. As linhas 251 a 256 posicionam o **secondFloorLightPanel** acima do **secondFloorButtonPanel**. As linhas 259 a 269 instanciam o **AnimatedPanel doorPanel**, posicionam o mesmo em relação à posição do **elevatorPanel** (porque o **elevatorPanel** irá conter o **doorPanel**) e fazem a atribuição de sequências de *frames* descrevendo a animação da porta se abrindo e fechando. As linhas 272 e 273 instanciam o **AnimatedPanel elevatorLightPanel** e o posicionam sobre o **elevatorPanel** e à esquerda do **doorPanel**. As linhas 276 a 283 instanciam o **bellPanel**, posicionam no abaixo do **elevatorLightPanel** e, depois, atribuem uma sequência de *frames* descrevendo a animação do **Button** pressionado. Finalmente, a linha 297 instancia a **ArrayList** que guarda os **AnimatedPanels** associados com as **Persons** no modelo.

Depois que o construtor **ElevatorView** chamou o método **instantiatePanels**, o construtor chama o método **placePanelsOnView** (linhas 302 a 322), que adiciona todos os **Panels** instanciados à **ElevatorView**. As linhas 317 a 320 adicionam **doorPanel**, **elevatorLightPanel**, **bellPanel** e **elevatorButtonPanel** ao **elevatorPanel**. O construtor da **ElevatorView** então chama o método **initializeAudio** (linhas 325 a 342). As linhas 328 e 329 instanciam um objeto **SoundEffects** e as linhas 331 a 336 usam o método **getAudioClip** do objeto **SoundEffects** para devolver os **AudioClips** para a simulação. O método **play** da classe **AudioClip** reproduz o **AudioClip** – a **ElevatorView** usa este método para sons que não são repetidos, como o toque da **Bell**. O método **loop** da classe **AudioClip** reproduz o clipe continuamente – a **ElevatorView** usa este método para sons que são repetidos, como o som de passos. As linhas 339 e 340 instanciam o objeto **ElevatorMusic** e asseguram que os dados MIDI são válidos.

Finalmente, as linhas 162 e 163 (no construtor **ElevatorView**) calculam a distância entre os dois **Floors** (i. e., a distância que o **elevatorPanel** vai percorrer). As linhas 166 a 169 usam a equação $taxa = distância / tempo$ para determinar a velocidade do **elevatorPanel** ao se mover. Finalmente, a linha 172 chama o método **startAnimation**, que dispara o **timer** da animação.

A **ElevatorView** anima os **ImagePanels** usando **animationTimer** (linha 142), uma interface da classe **javax.swing.Timer**. O **animationTimer** é disparado no construtor de **ElevatorView** através do método **startAnimation** (linhas 345 a 356). A classe **ElevatorView** implementa a interface **ActionListener** para esperar por **ActionEvents**. O **animationTimer** envia um **ActionEvent** para a **ElevatorView** a cada 50 (**ANIMATION_DELAY**) milissegundos. Quando a **ElevatorView** recebe um **ActionEvent**, a **ElevatorView** chama o método **actionPerformed** (linhas 365 a 385). A linha 367 neste método atualiza a posição e a imagem atual do **elevatorPanel** e dos filhos do **elevatorPanel**. As linhas 369 e 370 permitem que

o **firstFloorButtonPanel** e o **secondFloorButtonPanel** atualizam a si mesmos. As linhas 374 a 381 percorrem a **List personAnimatedPanels** e atualizam a posição e a imagem atual de cada **AnimatedPanel** associado com uma **Person** no modelo. Finalmente, a linha 383 chama o método **repaint** para redesenhar na tela todos os **ImagePanels** adicionados à **ElevatorView**.

Apresentamos um diagrama de objetos que lista todos os objetos na **ElevatorView**. Lembre-se de que um diagrama de objetos fornece uma fotografia instantânea da estrutura quando o sistema está sendo executado. O diagrama de objetos da Fig. I.4 representa a **ElevatorView** depois de invocar o construtor.

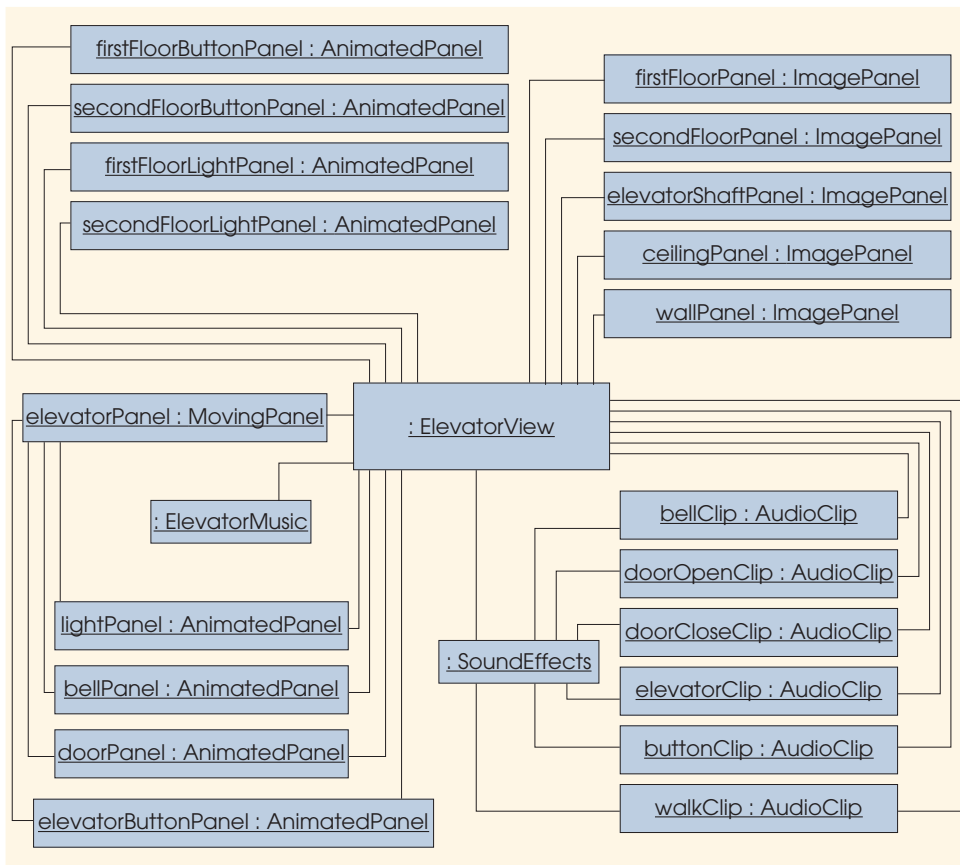


Fig I.4 Diagrama de objetos para a **ElevatorView** após a inicialização.

O objeto **ElevatorView** está vinculado (contém uma associação) com todos os objetos apresentados na Fig. I.4. O **elevatorPanel** tem ligações com os objetos **elevatorLightPanel**, **bellPanel**, **doorPanel** e **elevatorButtonPanel**. Esta associação fornece uma visualização do que está acontecendo no modelo – o **Elevator** contém uma **Light**, uma **Bell**, uma **Door** e um **Button**. O objeto **SoundEffects** está ligado com os objetos **AudioClip**, porque o objeto **SoundEffects** gera os objetos **AudioClip**.

I.5 Tratamento de eventos

A Fig. 13.19 especificou que a **ElevatorView** implementa a interface **ElevatorModelListener**, a qual implementa todas as interfaces na simulação. A **ElevatorSimulation** registra a **ElevatorView** como um ouvinte (*listener*) para eventos do **ElevatorModel**; em outras palavras, o **ElevatorModel** envia todos os eventos gerados no modelo para a **ElevatorView**.

Todo o método que implementa uma interface recebe um objeto de evento do tipo **ElevatorModelEvent** (ou uma subclasse) como parâmetro. Por exemplo, o método **doorOpened** recebe um **DoorEvent**. O Apêndice G contém referência adicional sobre eventos e *listeners*. As reações seguintes analisam os tipos de eventos que a **ElevatorView** trata.

1.5.1 Tipos de eventos ElevatorMoveEvent

O **ElevatorModel** envia um **ElevatorMoveEvent** quando o **Elevator** partiu ou chegou no modelo. O **ElevatorModel** invoca o método **elevatorDeparted** (linhas 435 a 493) quando o **Elevator** partiu de um **Floor**. As linhas 441 a 475 determinam se um **AnimatedPanel** associado com uma **Person** está sobreposto ao **elevatorPanel** percorrendo **personAnimatedPanels** e testando se qualquer **AnimatedPanel** na **List** tem uma coordenada *x* na tela maior do que aquela do **elevatorPanel**. Se este for o caso, a **Person** está dentro do **Elevator** e as linhas 465 a 468 adicionam o **AnimatedPanel** associado com aquela **Person** ao **elevatorPanel**. Independentemente da **Person** estar ou não no **Elevator**, as linhas 478 a 515 ajustam a velocidade do **elevatorPanel** de acordo com a direção em que o **Elevator** deve se mover. A linha 500 reproduz a **elevatorMusic**.

O **ElevatorModel** invoca o método **elevatorArrived** (linhas 496 a 515) quando o **Elevator** chegou em um **Floor**. A linha 499 faz parar o **elevatorPanel** e a linha 512 faz parar a **elevatorMusic**. As linhas 506 a 513 mudam a direção do **elevatorPanel** para a próxima viagem.

1.5.2 Tipos de eventos PersonMoveEvent

O **ElevatorModel** envia um **PersonMoveEvent** quando uma **Person** executou alguma ação no modelo que a **ElevatorView** deve representar. O **ElevatorModel** invoca o método **personCreated** (linhas 518 a 582) quando o modelo instancia uma nova **Person**. As linhas 526 e 527 instanciam um **AnimatedPanel** para uma **Person**. As linhas 531 e 545 determinam em que **Floor** situar o **AnimatedPanel**, dependendo do **Floor** no qual o evento foi gerado. As linhas 548 a 553 adicionam ao **AnimatedPanel** seqüências de *frames* que descrevem a pessoa caminhando e pressionando um **Button**. As linhas 556 a 572 animam a **Person** caminhando, determinam a velocidade necessária da **Person** para chegar ao **Button** no andar e reproduzem o efeito sonoro de passos. Finalmente, as linhas 575 a 580 adicionam o **AnimatedPanel** associado com a **Person** à **List personAnimatedPanels**, usando um bloco **synchronized** (linhas 575 a 578) para garantir que nenhum outro objeto possa acessar a **List**.

O **ElevatorModel** invoca o método **personArrived** (linhas 585 a 604) quando uma **Person** chegou ao **Elevator**. A linha 588 chama o método **getPersonPanel** (linhas 414 a 432), que determina o **AnimatedPanel** associado com a **Person** que emitiu o evento. Especificamente, o método **getPersonPanel** percorre a **List personAnimatedPanels** e devolve o **AnimatedPanel** cujo identificador coincide com o identificador do **PersonMoveEvent**. As linhas 590 a 603 no método **personArrived** fazem este **AnimatedPanel** parar de se mover. A linha 596 faz parar o som de passos chamando o método **stopWalkingSound** (linhas 397 a 411), que faz parar o **AudioClip** que está reproduzindo o som de passos se nenhuma **Person** estiver caminhando.

O **ElevatorModel** invoca o método **personPressedButton** (linhas 607 a 623) quando uma **Person** pressionou um **Button**. A linha 610 determina qual o **AnimatedPanel** associado com a **Person** que pressionou o **Button**. A linha 616 chama o método **playAnimation**, que reproduz a seqüência de animação daquela **Person** pressionando o **Button**.

O **ElevatorModel** invoca o método **personEntered** (linhas 626 a 647) quando uma **Person** está para entrar no **Elevator**. A linha 629 recupera o **AnimatedPanel** associado com a **Person** entrando no **Elevator**. As linhas 634 a 640 determinam a velocidade necessária para entrar no **Elevator**. As linhas 643 a 645 animam este **AnimatedPanel** para caminhar no **elevatorPanel**.

O **ElevatorModel** invoca o método **personDeparted** (linhas 650 a 697) quando uma **Person** está para sair do **Elevator**. A linha 653 determina o **AnimatedPanel** associado com a **Person** saindo do **Elevator**. As linhas 658 a 661 determinam a velocidade necessária para aquela **Person** caminhar através do **Floor** para sair da simulação. As linhas 664 a 688 posicionam o **AnimatedPanel** associado com a **Person** no **Floor** em frente do **Elevator** removendo o **AnimatedPanel** do **elevatorPanel** e adicionando o **AnimatedPanel** à **ElevatorView**. As linhas 691 a 695 animam este **AnimatedPanel** para caminhar através do **firstFloorPanel** ou do **secondFloorPanel** e iniciam o som de passos.

O **ElevatorModel** invoca o método **personExited** (linhas 700 a 718) quando uma **Person** saiu da simulação. A linha 703 determina o **AnimatedPanel** associado com a **Person** que saiu da simulação. As linhas 711 a 716 removem da **ElevatorView** o **AnimatedPanel** associado com aquela **Person** e fazem parar o som de passos.

I.5.3 Tipos de eventos DoorEvent

O **ElevatorModel** envia um **DoorEvent** para a **ElevatorView** quando uma **Door** abriu ou fechou no modelo. O **ElevatorModel** invoca o método **doorOpened** (linhas 721 a 637) quando uma **Door** abriu. As linhas 724 a 730 animam o **doorPanel** se abrindo e as linhas 733 e 734 reproduzem o **doorOpenClip**, que é o efeito de som associado com a abertura da porta.

O **ElevatorModel** invoca o método **doorClosed** (linhas 739 a 754) quando uma **Door** fechou. As linhas 742 a 748 animam o **doorPanel** fechando e as linhas 751 e 752 reproduzem o **doorClosedClip**, que é o efeito de som associado com o fechamento da **Door**.

I.5.4 Tipos de eventos ButtonEvent

O **ElevatorModel** envia um **ButtonEvent** para a **ElevatorView** quando um **Button** foi pressionado ou desligado no modelo. O **ElevatorModel** invoca o método **buttonPressed** (linhas 757 a 786) quando um **Button** foi pressionado. As linhas 760 a 761 determinam a **Location** em que o **Button** foi pressionado. Se a **Location** é o **Elevator**, então as linhas 764 a 767 reproduzem a animação de **Button** pressionado dentro do **Elevator**. Se a **Location** for o primeiro **Floor**, as linhas 772 a 775 reproduzem a animação do **Button** pressionado no primeiro **Floor**. Se a **Location** for o segundo **Floor**, as linhas 778 a 781 reproduzem a animação do **Button** sendo pressionado no segundo **Floor**. As linhas 783 e 784 reproduzem o **buttonClip**, que é o efeito de som associado com o **Button** sendo pressionado.

O **ElevatorModel** invoca o método **buttonReset** (linhas 789 a 829) quando um **Button** foi desligado. As linhas 792 e 793 determinam a **Location** em que o **Button** foi desligado. Se a **Location** for o **Elevator**, as linhas 796 a 803 mudam a imagem do **elevatorButtonPanel** para aquela do **Button** desligado. Se a **Location** for o primeiro **Floor**, as linhas 808 a 816 mudam a imagem do **firstFloorButtonPanel** associada como o **Button** desligado. Se a **Location** for o segundo **Floor**, as linhas 819 a 827 mudam a imagem do **secondFloorButtonPanel** associada com o **Button** desligado.

I.5.5 Tipos de eventos BellEvent

O **ElevatorModel** envia um **BellEvent** para a **ElevatorView** invocando o método **bellRang** (linhas 832 a 838) quando uma **Bell** tocou no modelo. A linha 834 anima o **bellPanel** e as linhas 836 e 837 reproduzem o **bellClip**, que é o efeito de som associado com a **Bell** tocando.

I.5.6 Tipos de eventos LightEvent

O **ElevatorModel** envia um **LightEvent** para a **ElevatorView** quando uma **Light** mudou de estado no modelo. O **ElevatorModel** invoca o método **lightTurnedOn** (linhas 841 a 858) quando uma **Light** acende. A linha 844 acende o **elevatorLightPanel**. As linhas 846 a 856 determinam em que **Floor** a **Light** acendeu e, então, iluminam o **AnimatedPanel** associado com aquela **Light** na **ElevatorView**. O **ElevatorModel** invoca o método **lightTurnedOff** (linhas 861 a 878) quando uma **Light** apaga. A linha 864 apaga o **elevatorLightPanel**. As linhas 866 a 876 determinam em que **Floor** a **Light** foi apagada e, então, desligam o **AnimatedPanel** associado com aquela **Light** na **ElevatorView**.

I.6 Diagrama de componentes revisitado

Na Seção 13.17, apresentamos o diagrama de componentes para a simulação do elevador e, no Apêndice G e no Apêndice H, adicionamos componentes aos pacotes **event** e **model**, respectivamente. A Fig. I.5 apresenta o diagrama de componentes para o pacote **view**, que contém os componentes **ElevatorView.java**, **ImagePanel.java**, **MovingPanel.java**, **AnimatedPanel.java**, **ElevatorMusic.java** e **SoundEf-**

`fects.java`, `ElevatorView.java` agrega os pacotes `images`, `sounds` e `event`. Os pacotes `images` e `sounds` contêm todos os arquivos de imagem e som (componentes) usados por `ElevatorView.java`, respectivamente. O diagrama não mostra os componentes destes diretórios, porque existe uma quantidade muito grande de arquivos de gráficos e de áudio para representar em uma página – o conteúdo destes pacotes pode ser encontrado nas estruturas de diretório

```
com/deitel/jhttp4/elevator/view/images
```

```
com/deitel/jhttp4/elevator/view/sounds
```

(i. e., nos diretórios `images` e `sounds` nos quais as classes para a visão estão localizadas no sistema de arquivos).

I.7 Conclusão

Parabéns! Você completou um estudo de caso de OOD/UML de “porte empresarial”. Você está bem preparado para enfrentar problemas de projeto mais substanciais e prosseguir para estudo mais aprofundado de OOD com a UML. Acreditamos que você tenha desenvolvido uma maior valorização e compreensão dos processos de projeto e implementação. Agora, você pode usar Java para implementar projetos de sistemas substanciais orientados a objetos, gerados pela UML. Esperamos que você tenha gostado de usar Java e a UML para construir este estudo de caso ao mesmo tempo em que aprendia os recursos que as duas tecnologias têm a oferecer. Além disso, esperamos que você tenha gostado de usar os recursos para GUI, gráficos e sons de Java, ao mesmo tempo em que aprendia conceitos importantes relacionados à orientação a objetos e relacionados com Java, como classes, objetos, construção de GUI, herança, tratamento de eventos e *multithreading*.

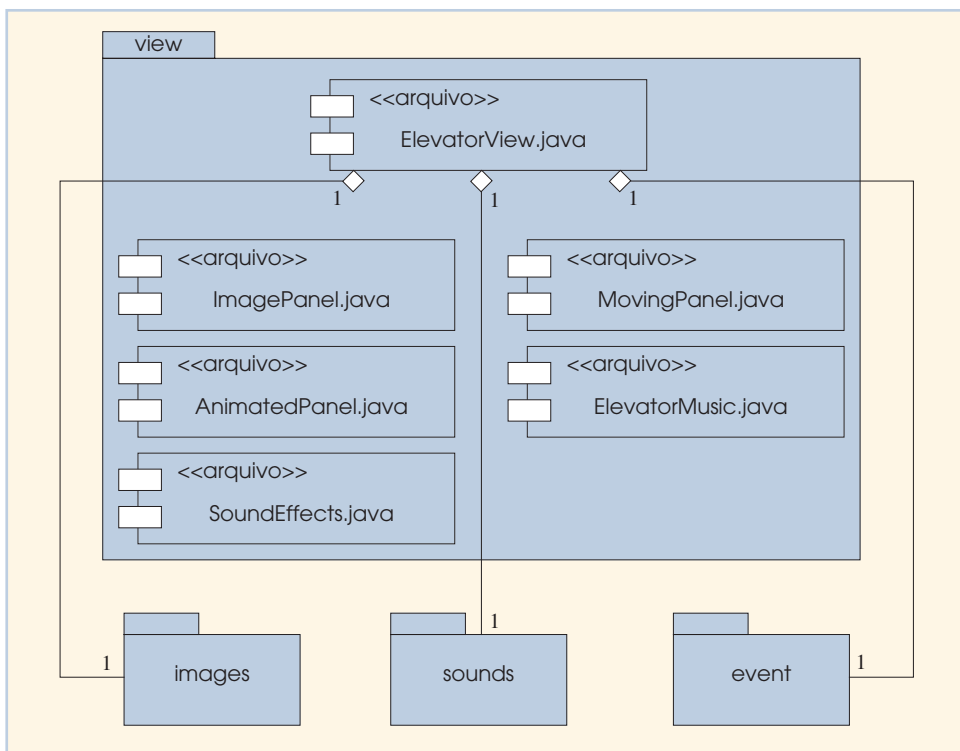


Fig I.5 Diagrama de componentes para o pacote `view`.