



Eventos e interfaces *listener* do elevador (no CD)

G.1 Introdução

Na Seção 10.22 de “Pensando em objetos”, discutimos como funciona o tratamento de eventos em nossa simulação de elevador. Mencionamos que, para um objeto receber um evento, ele precisa registrar um ouvinte (*listener*) para aquele evento. Portanto, a classe desse objeto precisa implementar uma interface *listener* apropriada que contenha métodos que recebem um objeto evento como parâmetro. Nesta seção, apresentamos os eventos e as interfaces *listener* usados em nossa simulação.

G.2 Eventos

As próximas oito figuras (Fig. G.1 a Fig. G.7) contêm os eventos do sistema. Cada evento herda da classe **ElevatorModelEvent** na Fig. G.1. Esta classe contém uma referência **Location** (linha 11) que representa onde o evento foi gerado – em nossa simulação, esta referência é o objeto **Elevator** ou o objeto **Floor**. A classe **ElevatorModelEvent** também contém uma referência **Object** (linha 14) que representa a fonte que gerou o evento. Os métodos **getLocation** (linhas 30 a 33) e **getSource** (linhas 42 a 45) devolvem as referências **Location** e **Object**, respectivamente. Observe que cada subclasse de **ElevatorModelEvent** (Fig. G.2 a Fig. G.7) fornece somente um construtor que chama o construtor da classe **ElevatorModelEvent**. Como mencionamos na Seção 10.22, dividir a classe **ElevatorModelEvent** em diversas subclasses de evento facilita a compreensão do tratamento de eventos em nossa simulação.

```
1 // ElevatorModelEvent.java
2 // Pacote básico de eventos na simulação de elevador
3 package com.deitel.jhtp4.elevator.event;
4
5 // Pacotes Deitel
6 import com.deitel.jhtp4.elevator.model.*;
7
8 public class ElevatorModelEvent {
9
10     // Location onde o ElevatorModelEvent foi gerado
11     private Location location;
12
13     // Object de origem que gerou o ElevatorModelEvent
```

Fig. G.1 Superclasse **ElevatorModelEvent** para eventos no modelo da simulação de elevador. (parte 1 de 2).

```

14     private Object source;
15
16     // construtor ElevatorModelEvent configura Location
17     public ElevatorModelEvent( Object source, Location location )
18     {
19         setSource( source );
20         setLocation( location );
21     }
22
23     // configura a Location de ElevatorModelEvent
24     public void setLocation( Location eventLocation )
25     {
26         location = eventLocation;
27     }
28
29     // obtém a Location de ElevatorModelEvent
30     public Location getLocation()
31     {
32         return location;
33     }
34
35     // configura a origem de ElevatorModelEvent
36     private void setSource( Object eventSource )
37     {
38         source = eventSource;
39     }
40
41     // obtém a origem de ElevatorModelEvent
42     public Object getSource()
43     {
44         return source;
45     }
46 }

```

Fig. G.1 Superclasse `ElevatorModelEvent` para eventos no modelo da simulação de elevador. (parte 2 de 2).

```

1  // BellEvent.java
2  // Indica que a Bell tocou
3  package com.deitel.jhttp4.elevator.event;
4
5  // Pacotes Deitel
6  import com.deitel.jhttp4.elevator.model*;
7
8  public class BellEvent extends ElevatorModelEvent {
9
10     // construtor BellEvent
11     public BellEvent( Object source, Location location )
12     {
13         super( source, location );
14     }
15 }

```

Fig. G.2 Subclasse `BellEvent` de `ElevatorModelEvent` indicando que a Bell tocou.

```

1 // ButtonEvent.java
2 // Indica que um Button mudou de estado
3 package com.deitel.jhtp4.elevator.event;
4
5 // Pacotes Deitel
6 import com.deitel.jhtp4.elevator.model.*;
7
8 public class ButtonEvent extends ElevatorModelEvent {
9
10     // construtor ButtonEvent
11     public ButtonEvent( Object source, Location location )
12     {
13         super( source, location );
14     }
15 }

```

Fig. G.3 Subclasse `ButtonEvent` de `ElevatorModelEvent` indicando que um `Button` mudou de estado.

```

1 // DoorEvent.java
2 // Indica que uma Door mudou de estado
3 package com.deitel.jhtp4.elevator.event;
4
5 // Pacotes Deitel
6 import com.deitel.jhtp4.elevator.model.*;
7
8 public class DoorEvent extends ElevatorModelEvent {
9
10     // construtor DoorEvent
11     public DoorEvent( Object source, Location location )
12     {
13         super( source, location );
14     }
15 }

```

Fig. G.4 Subclasse `DoorEvent` de `ElevatorModelEvent` indicando que uma `Door` mudou de estado.

```

1 // ElevatorMoveEvent.java
2 // Indica em que Floor o Elevator chegou ou do qual partiu
3 package com.deitel.jhtp4.elevator.event;
4
5 // Pacotes Deitel
6 import com.deitel.jhtp4.elevator.model.*;
7
8 public class ElevatorMoveEvent extends ElevatorModelEvent {
9
10     // construtor ElevatorMoveEvent
11     public ElevatorMoveEvent( Object source, Location location )
12     {
13         super( source, location );
14     }
15 }

```

Fig. G.5 Subclasse `ElevatorMoveEvent` de `ElevatorModelEvent` indicando em que `Floor` o `Elevator` chegou ou do qual ele partiu.

```

1  // LightEvent.java
2  // Indica em que Floor a Light mudou de estado
3  package com.deitel.jhttp4.elevator.event;
4
5  // Pacotes Deitel
6  import com.deitel.jhttp4.elevator.model.*;
7
8  public class LightEvent extends ElevatorModelEvent {
9
10     // construtor LightEvent
11     public LightEvent( Object source, Location location )
12     {
13         super( source, location );
14     }
15 }

```

Fig. G.6 Subclasse `LightEvent` de `ElevatorModelEvent` indicando qual o Floor cuja `Light` mudou de estado.

A classe `PersonMoveEvent` (Fig. G.7) tem uma estrutura ligeiramente diferente daquela das outras classes de evento. A linha 11 declara o atributo `int ID`. Descobriremos no Apêndice I que a `ElevatorView` obtém este atributo através do método `getID` (linhas 22 a 25) para determinar qual `Person` enviou o evento.

```

1  // PersonMoveEvent.java
2  // Indica que uma Person se moveu
3  package com.deitel.jhttp4.elevator.event;
4
5  // Pacotes Deitel
6  import com.deitel.jhttp4.elevator.model.*;
7
8  public class PersonMoveEvent extends ElevatorModelEvent {
9
10     // identificador da Person que está enviando o Event
11     private int ID;
12
13     // construtor PersonMoveEvent
14     public PersonMoveEvent( Object source, Location location,
15         int identifier )
16     {
17         super( source, location );
18         ID = identifier;
19     }
20
21     // devolve identificador
22     public int getID()
23     {
24         return( ID );
25     }
26 }

```

Fig. G.7 Subclasse `PersonMoveEvent` de `ElevatorModelEvent` indicando que uma `Person` se moveu.

G.3 Listeners

As próximas oito figuras (Fig. G.8 a Fig. G.14) contêm as interfaces *listener* para a simulação do elevador. `BellListener` (Fig. G.8) fornece o método `bellRang` (linha 8), que é invocado quando a `Bell` toca. `ButtonListener` (Fig. G.9) fornece os métodos `buttonPressed` (linha 8) e `buttonReset` (linha 11), que esperam quando um

Button é pressionado ou desligado. **DoorListener** (Fig. G.10) fornece os métodos **doorOpened** (linha 8) e **doorClosed** (linha 11), que esperam pela abertura ou pelo fechamento de uma **Door**. **ElevatorMoveListener** (Fig. G.11) fornece os métodos **elevatorDeparted** (linha 8) e **elevatorArrived** (linha 11), que esperam as partidas e as chegadas do **Elevator**. **LightListener** (Fig. G.12) fornece os métodos **lightTurnedOn** (linha 8) e **lightTurnedOff** (linha 11) que esperam as mudanças de estado de **Light**. **PersonMoveListener** (Fig. G.13) fornece os métodos **personCreated** (linha 8), **personArrived** (linha 11), **personDeparted** (linha 14), **personPressedButton** (linhas 17 e 18), **personEntered** (linhas 21) e **personExited** (linha 24). Estes métodos esperam o momento durante o qual uma pessoa foi criada, chegou no **Elevator**, ou saiu dele, pressionou um **Button**, entrou no **Elevator** ou saiu da simulação, respectivamente. Finalmente, **ElevatorModelListener** (Fig. g.14) herda comportamentos de todas as interfaces *listener*. A **ElevatorView** utiliza a interface **ElevatorModelListener** na Seção 13.17 e no Apêndice I para receber eventos do **ElevatorModel**.

```

1 // BellListener.java
2 // Método invocado quando a Bell tocou
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface BellListener {
6
7     // invocado quando Bell tocou
8     public void bellRang( BellEvent bellEvent );
9 }

```

Fig. G.8 Método da interface **BellListener** para quando a **Bell** tocou.

```

1 // ButtonListener.java
2 // Método invocado quando Button foi pressionado ou desligado
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface ButtonListener {
6
7     // invocado quando Button foi pressionado
8     public void buttonPressed( ButtonEvent buttonEvent );
9
10    // invocado quando Button foi desligado
11    public void buttonReset( ButtonEvent buttonEvent );
12 }

```

Fig. G.9 Métodos da interface **ButtonListener** para quando o **Button** foi pressionado ou desligado.

```

1 // DoorListener.java
2 // Métodos invocados quando Door foi aberta ou fechada
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface DoorListener {
6
7     // invocado quando a Door abriu
8     public void doorOpened( DoorEvent doorEvent );
9
10    // invocado quando a Door fechou
11    public void doorClosed( DoorEvent doorEvent );
12 }

```

Fig. G.10 Métodos da interface **DoorListener** para quando a **Door** abriu ou fechou.

```

1 // ElevatorMoveListener.java
2 // Métodos invocados quando Elevator chegou ou partiu
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface ElevatorMoveListener {
6
7     // invocado quando Elevator partiu
8     public void elevatorDeparted( ElevatorMoveEvent moveEvent );
9
10    // invocado quando Elevator chegou
11    public void elevatorArrived( ElevatorMoveEvent moveEvent );
12 }

```

Fig. G.11 Métodos da interface `ElevatorMoveListener` para quando o Elevator partiu de um Floor ou chegou em um.

```

1 // LightListener.java
2 // Métodos invocados quando a Light ligou ou desligou
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface LightListener {
6
7     // invocado quando a Light ligou
8     public void lightTurnedOn( LightEvent lightEvent );
9
10    // invocado quando a Light desligou
11    public void lightTurnedOff( LightEvent lightEvent );
12 }

```

Fig. G.12 Método da interface `LightListener` para quando a Light ligou ou desligou.

```

1 // PersonMoveListener.java
2 // Métodos invocados quando a Person se moveu
3 package com.deitel.jhtp4.elevator.event;
4
5 public interface PersonMoveListener {
6
7     // invocado quando a Person foi instanciada no modelo
8     public void personCreated( PersonMoveEvent moveEvent );
9
10    // invocado quando a Person chegou no elevador
11    public void personArrived( PersonMoveEvent moveEvent );
12
13    // invocado quando a Person saiu do elevador
14    public void personDeparted( PersonMoveEvent moveEvent );
15
16    // invocado quando a Person pressionou o Button
17    public void personPressedButton(
18        PersonMoveEvent moveEvent );
19
20    // invocado quando a Person entrou no Elevator
21    public void personEntered( PersonMoveEvent moveEvent );
22
23    // invocado quando a Person saiu da simulação
24    public void personExited( PersonMoveEvent moveEvent );
25 }

```

Fig. G.13 Métodos da interface `PersonMoveListener` para quando a Person se moveu.

```

1 // ElevatorModelListener.java
2 // Listener para ElevatorView a partir de ElevatorModel
3 package com.deitel.jhttp4.elevator.event;
4
5 // ElevatorModelListener herda todas as interfaces Listener
6 public interface ElevatorModelListener extends BellListener,
7     ButtonListener, DoorListener, ElevatorMoveListener,
8     LightListener, PersonMoveListener {
9 }

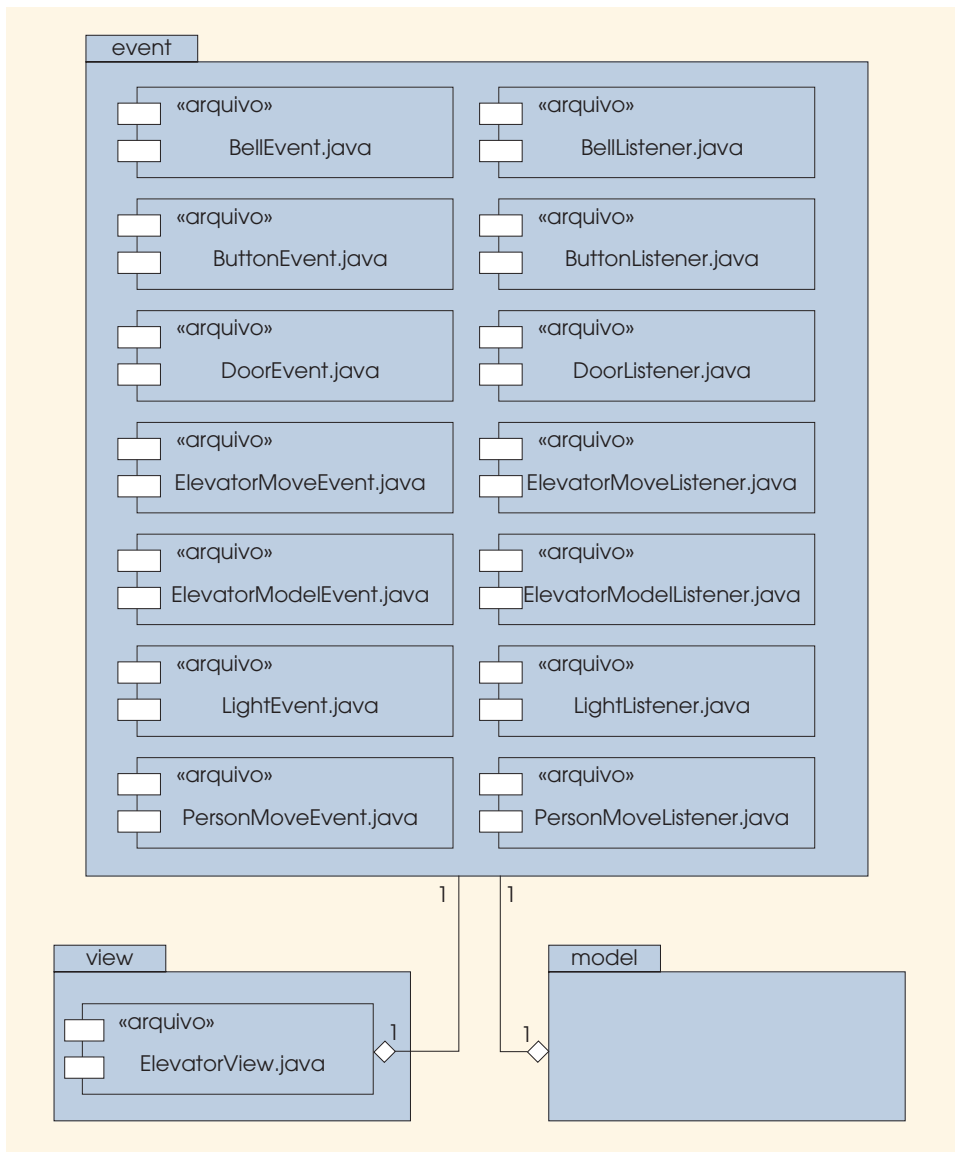
```

Fig. G.14 A interface `ElevatorModelListener` permite que o modelo envie todos os eventos para a visão.

G.4 Diagramas de componentes revisitados

Na Seção 13.17, apresentamos o diagrama de componentes para a simulação do elevador. Em nossa simulação, a `ElevatorView` e todos os objetos no modelo importam o pacote `event`. A Fig. G.15 apresenta o diagrama de componentes para o pacote `event`. Cada componente no pacote `event` é mapeado para uma classe da Fig. G.1 à Fig. G.14. De acordo com o diagrama de componentes, `ElevatorView.java` do pacote `view` agrega o pacote `event`. Em Java, esta agregação se traduz na classe `ElevatorView` importando o pacote `event`. Também de acordo com a Fig. G.15, o pacote `model` agrega o pacote `event` – i.e., cada componente no pacote `model` contém uma agregação com todos os componentes no pacote `event` (mostramos todos os componentes do pacote `model` em um diagrama de componentes separado, no Apêndice H). Em Java, esta agregação se traduz para cada classe no pacote `model` que importa o pacote `event`.

Isto conclui o apêndice sobre os eventos e as interfaces *listener* da simulação do elevador. Esperamos que você tenha achado esta referência muito útil para o material sobre o tratamento de eventos discutido na Seção 10.22 de “Pensando em objetos”. Nos próximos dois apêndices, implementamos o projeto para o modelo e a visão, e fornecemos os diagramas de componentes para os pacotes `model` e `view`.

**Fig. G.15** Diagrama de componentes para o pacote `event`.