```java
package juego;

import java.awt.Color;
import java.util.Random;

import entorno.Entorno;

public class Escaleras {

    private double x;
    private double y;
    private double ancho;
    private double alto;

    public Escaleras(int pos, Viga[] suelos) {

        Random rnd = new Random();
        boolean escaleraCompleta;
        this.ancho = 30;

        /*
         * Hay 2 tipos de escaleras. Las obligatorias que deben estar completas y ocupan
         * las "pos" del 0 al 4. Luego las adicionales que pueden estar imcompletas o
         * no. Ocupan las "pos" del 5 al 9.
         */

        // Para las obligatorias
        if (pos < 5) {

            int offsetEscalera = rnd.nextInt(50);

            if (pos % 2 == 0) {

                this.x = suelos[pos + 1].extremoDerecho() - 30 - offsetEscalera;

            } else {

                this.x = suelos[pos + 1].extremoIzquierdo() + 30 + offsetEscalera;
            }

            // El punto mediatriz del segmento que representa la distancia entre la
            // superficie superior de la viga que funciona
            // como el suelo, y la superficie superior de la viga siguiente. Dicho punto es
            // el centro Y de la escalera.
            this.y = ((suelos[pos].dondeEmpiezaElSuelo() - suelos[pos + 1].dondeEmpiezaElSuelo()) / 2)
                    + suelos[pos + 1].dondeEmpiezaElSuelo();

            // El alto es la distancia entre la superficie superior de la viga que funciona
            // como el suelo, y la superficie superior de la viga siguiente.
            this.alto = suelos[pos].dondeEmpiezaElSuelo() - suelos[pos + 1].dondeEmpiezaElSuelo();

        }

        // Para las escaleras adicionales
        else {

```

```
57              int offsetEscalera = rnd.nextInt(150);
58
59              // Las escaleras pares, conectan con vigas superiores que no tocan
                el extremo
60              // derecho.
61              // Las escaleras impares, conectan con vigas superiores que no tocan
                el extremo
62              // izquierdo.
63              // A su vez, la posición de la escalera, está aumentada 4 veces con
                repecto al
64              // indice de la viga superior
65              // con la cual debe conectar.
66              if (pos % 2 == 0) {
67
68                  this.x = suelos[pos - 4].extremoDerecho() - 250 - offsetEscalera;
69
70              } else {
71
72                  this.x = suelos[pos - 4].extremoIzquierdo() + 250 +
                    offsetEscalera;
73              }
74
75              // Para todas las escaleras adicionales, expecto la de planta baja.
76              if (pos != 5) {
77
78                  // Generamos un random que ayudará a elegir si la escalera se
                    presentará de
79                  // forma completa o no
80                  int eleccionEscaleraCompleta = rnd.nextInt(300);
81
82                  // De esta forma existe sólo un 33% de posibilidades de que
                    aparezca completa.
83                  if (eleccionEscaleraCompleta % 3 == 0) {
84
85                      escaleraCompleta = true;
86                  } else {
87                      escaleraCompleta = false;
88                  }
89
90                  // La escalera adicional de planta baja, nunca estará completa
91              } else {
92                  escaleraCompleta = false;
93              }
94
95              if (escaleraCompleta) {
96                  // Método normal para calcular la el centro "Y" y la altura.
97                  this.y = ((suelos[pos - 5].dondeEmpiezaElSuelo() - suelos[pos - 4
                    ].dondeEmpiezaElSuelo()) / 2)
98                          + suelos[pos - 4].dondeEmpiezaElSuelo();
99
100                 this.alto = suelos[pos - 5].dondeEmpiezaElSuelo() - suelos[pos -
                    4].dondeEmpiezaElSuelo();
101
102             }
103
104             else {
105
106                 // Al método normal le corremos 25 pixeles hacia abajo y la
                    altura es la mitad.
107                 // impidiendo que la escalera esté completa
108                 this.y = ((suelos[pos - 5].dondeEmpiezaElSuelo() - suelos[pos - 4
                    ].dondeEmpiezaElSuelo()) / 2)
109                         + suelos[pos - 4].dondeEmpiezaElSuelo() + 25;
```

```
110
111                 this.alto = (suelos[pos - 5].dondeEmpiezaElSuelo() - suelos[pos -
                     4].dondeEmpiezaElSuelo()) / 2;
112
113             }
114
115         }
116
117     }
118
119     public void dibujar(Entorno entorno) {
120
121         // Rectángulo básico de la viga, respetando los valores indicados por el
122         // constructor
123         entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0.0,
            Color.BLUE);
124
125         double paso = this.y + (this.alto / 2) - 3;
126
127         // Se decide que la suma de la base de un triangulo, la punta del
            triángulo
128         // adyacente y un espacio
129         // extra sea la 25ava parte del ancho de la viga - 4 pixeles
130         double rectangulos = (this.alto / 10);
131
132         // Indica la cantidad de parejas de triangulos dibujados. Una pareja es un
133         // triangulo con la punta hacia arriba
134         // y el otro con la punta hacia abajo.
135         int dibujados = 0;
136
137         // Este bucle dibuja la pareja de triángulos a lo largo de la viga.
138
139         while (dibujados <= rectangulos) {
140
141             entorno.dibujarRectangulo(this.x, paso, 28, 9, 0.0, java.awt.Color.
                BLACK);
142             paso -= 10;
143
144             dibujados += 1;
145
146         }
147
148     }
149
150     public int lateralDerecho() {
151         return (int) this.x + 15;
152     }
153
154     public int lateralIzquierdo() {
155         return (int) this.x - 15;
156     }
157
158     public int extremoSuperior() {
159         return (int) (this.y - (this.alto / 2));
160     }
161
162     public int extremoInferior() {
163         return (int) (this.y + (this.alto / 2));
164     }
165
166 }
167
```