



```
1 /**
2 * Eine Klasse, die die Posten in einer Auktion modelliert.
3 * Ein Posten kann ein einzelner Gegenstand oder eine
4 * Zusammenstellung von Gegenständen sein.
5 *
6 * @author David J. Barnes und Michael Kölling.
7 * @version 2016.02.29
8 */
9 public class Posten
10 {
11     // eine eindeutige Nummer zur Identifikation dieses Postens
12     private final int nummer;
13     // eine Beschreibung dieses Postens
14     private String beschreibung;
15     // das aktuell höchste Gebot für diesen Posten
16     private Gebot hoechstesGebot;
17
18     /**
19      * Erzeuge einen Posten, mit Nummer und Beschreibung.
20      * @param nummer      die Postennummer
21      * @param beschreibung eine Beschreibung dieses Postens
22      */
23     public Posten(int nummer, String beschreibung)
24     {
25         this.nummer = nummer;
26         this.beschreibung = beschreibung;
27         this.hoechstesGebot = null;
28     }
29
30     /**
31      * Ein Versuch, für diesen Posten zu bieten. Ein erfolgreiches
32      * Gebot muss höher sein als alle bisherigen Gebote.
33      * @param gebot  ein neues Gebot
34      * @return true falls erfolgreich, false sonst
35      */
36     public boolean hoeheresGebot(Gebot gebot)
37     {
38         if(hoechstesGebot == null) {
39             // es gibt kein älteres Gebot
40             hoechstesGebot = gebot;
41             return true;
42         }
43         else if (gebot.gibHoehe() > hoechstesGebot.gibHoehe()){
44             // das Gebot ist besser als das letzte
45             hoechstesGebot = gebot;
46             return true;
47         }
48         else {
49             // das Gebot ist nicht besser
50             return false;
51         }
52     }
53
54     /**
```

```
55 * @return eine String-Darstellung der Details dieses Postens
56 */
57 public String toString()
58 {
59     String details = nummer + ":" + beschreibung;
60     if(hoechstesGebot != null) {
61         details += "    Gebot: " +
62                     hoechstesGebot.gibHoehe() +
63                     "    Name: " +
64                     hoechstesGebot.gibBieter().gibName();
65     }
66     else {
67         details += "    (Kein Gebot)";
68     }
69     return details;
70 }
71
72 /**
73 * @return die Nummer dieses Postens
74 */
75 public int gibNummer()
76 {
77     return nummer;
78 }
79
80 /**
81 * @return die Beschreibung dieses Postens
82 */
83 public String gibBeschreibung()
84 {
85     return beschreibung;
86 }
87
88 /**
89 * Liefere das höchste Gebot für diesen Posten.
90 * Das Ergebnis kann 'null' sein, wenn noch kein
91 * Gebot abgegeben wurde.
92 * @return das höchste Gebot
93 */
94 public Gebot gibHoechstesGebot()
95 {
96     return hoechstesGebot;
97 }
98 }
```

```
1 import java.util.ArrayList;
2
3 /**
4  * Ein einfaches Modell einer Auktion.
5  * Ein Exemplar dieser Klasse hält eine Liste von Posten,
6  * die beliebig lang werden kann.
7  *
8  * @author David J. Barnes und Michael Kölling
9  * @version 2016.02.29
10 */
11 public class Auktion
12 {
13     // eine Liste der Posten dieser Auktion
14     private ArrayList<Posten> postenliste;
15     // die Nummer, die an den nächsten Posten vergeben wird,
16     // der für diese Auktion angemeldet wird
17     private int naechstePostennummer;
18
19     /**
20      * Erzeuge eine neue Auktion.
21      */
22     public Auktion()
23     {
24         postenliste = new ArrayList<>();
25         naechstePostennummer = 1;
26     }
27
28     /**
29      * Melde einen Posten für diese Auktion an.
30      * @param beschreibung die Beschreibung des Postens
31      */
32     public void postenAnmelden(String beschreibung)
33     {
34         postenliste.add(new Posten(naechstePostennummer, beschreibung));
35         naechstePostennummer++;
36     }
37
38     /**
39      * Zeige die komplette Liste der Posten dieser Auktion.
40      */
41     public void zeigePostenliste()
42     {
43         for (Posten posten : postenliste) {
44             System.out.println(posten.toString());
45         }
46     }
47
48     /**
49      * Gebe für einen Posten ein Angebot ab.
50      * Es erfolgt eine Ausgabe, ob das Gebot erfolgreich war oder nicht.
51      * @param postennummer der Posten, für den geboten wird
52      * @param bieter die Person, die bietet
53      * @param betrag die Höhe des Gebots
54      */
```



```
108                     nummer);
109                 }
110             return gewaehlerPosten;
111         }
112     else {
113         System.out.println("Einen Posten mit der Nummer: " + nummer +
114                         " gibt es nicht.");
115         return null;
116     }
117 }
118 */
119 }
120
121 public void beenden()
122 {
123     for(Posten posten : postenliste)
124     {
125         System.out.println(posten.toString());
126         /*
127          * Weil wir eine toString-Methode haben, können wir uns das
128          * folgende sparen:
129          *
130         System.out.println(posten.gibBeschreibung());
131         if(posten.gibHoechstesGebot() != null)
132         {
133
134             System.out.println("Name: " +
135 posten.gibHoechstesGebot().gibBieter().gibName() +
136                     " , Höchstes Gebot: " + posten.gibHoechstesGebot().gibHoehe()
137 + " Euro.");
138         }
139     else
140     {
141         System.out.println("Kein Gebot vorhanden");
142     }
143     */
144 }
145
146 public ArrayList<Posten> gibUnverkaufte()
147 {
148     ArrayList<Posten> unverkauftePostenliste = new ArrayList<>();
149     for(Posten posten : postenliste)
150     {
151         if(posten.gibHoechstesGebot() == null)
152         {
153             unverkauftePostenliste.add(posten);
154         }
155     }
156     return unverkauftePostenliste;
157 }
158
159 public Posten entfernePosten(int nummer)
{
```

```
160     Posten posten = gibPosten(nummer);
161     postenliste.remove(posten);
162     return posten;
163 }
164
165 public ArrayList<Posten> entferneVerkaufte()
166 {
167     //todo!
168     return null;
169 }
170 }
171 }
```

```
1 /**
2  * Modellierung von Personen, die an einer
3  * Auktion teilnehmen.
4  *
5  * @author David J. Barnes und Michael Kölling.
6  * @version 2016.02.29
7  */
8 public class Person
9 {
10     // der Name dieser Person
11     private final String name;
12
13     /**
14      * Erzeuge eine neue Person mit dem angegebenen Namen.
15      * @param name  der Name der Person
16      */
17     public Person(String name)
18     {
19         this.name = name;
20     }
21
22     /**
23      * @return den Namen der Person
24      */
25     public String gibName()
26     {
27         return name;
28     }
29 }
30 }
```

```
1 /**
2  * Eine Klasse, die Gebote in einer Auktion modelliert.
3  * Ein Gebot enthält eine Referenz auf die Person, die
4  * das Gebot abgegeben hat, und die Höhe des Gebots.
5  *
6  * @author David J. Barnes und Michael Kölling.
7  * @version 2016.02.29
8 */
9 public class Gebot
10 {
11     // die Person, die das Gebot abgegeben hat.
12     private final Person bieter;
13     // Die Höhe des Gebots. Da dies potentiell ein sehr hohes
14     // Gebot sein kann, wurde der Basistyp 'long' gewählt.
15     private final long hoehe;
16
17 /**
18     * Erzeuge ein Gebot.
19     * @param bieter die Person, die das Gebot abgibt
20     * @param hoehe die Höhe des Gebots
21     */
22     public Gebot(Person bieter, long hoehe)
23     {
24         this.bieter = bieter;
25         this.hoehe = hoehe;
26     }
27
28 /**
29     * Liefere den Bieter dieses Gebots.
30     * @return den Bieter
31     */
32     public Person gibBieter()
33     {
34         return bieter;
35     }
36
37 /**
38     * Liefere die Höhe dieses Gebots.
39     * @return die Höhe dieses Gebots
40     */
41     public long gibHoehe()
42     {
43         return hoehe;
44     }
45 }
```

```
1
2
3 import static org.junit.Assert.*;
4 import org.junit.After;
5 import org.junit.Before;
6 import org.junit.Test;
7
8 /**
9  * Die Test-Klasse AuktionTest.
10 *
11 * @author (Ihr Name)
12 * @version (eine Versionsnummer oder ein Datum)
13 */
14 public class AuktionTest
15 {
16     private Auktion auktion1;
17     private Person person1;
18
19
20
21
22 /**
23  * Konstruktor fuer die Test-Klasse AuktionTest
24  */
25 public AuktionTest()
26 {
27 }
28
29 /**
30  * Setzt das Testgeruest fuer den Test.
31  *
32  * Wird vor jeder Testfall-Methode aufgerufen.
33  */
34 @Before
35 public void setUp()
36 {
37     auktion1 = new Auktion();
38     auktion1.postenAnmelden("Auto");
39     person1 = new Person("Felix");
40     auktion1.gibGebotAb(1, person1, 20);
41     auktion1.postenAnmelden("Fahrrad");
42     auktion1.postenAnmelden("Zelt");
43 }
44
45 /**
46  * Gibt das Testgeruest wieder frei.
47  *
48  * Wird nach jeder Testfall-Methode aufgerufen.
49  */
50 @After
51 public void tearDown()
52 {
53 }
```

```
55 |     @Test
56 |     public void testUnverkauftePosten()
57 |     {
58 |         java.util.ArrayList<Posten> arrayLis1 = auktion1.gibUnverkaufte();
59 |         assertEquals(2, arrayLis1.size());
60 |     }
61 |
62 |
63 | }
```

1 |