



```
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 BestandsverwalterTest.
10 *
11 * @author (Ihr Name)
12 * @version (eine Versionsnummer oder ein Datum)
13 */
14 public class BestandsverwalterTest
15 {
16     private Bestandsverwalter bestands1;
17     private Artikel artikel1;
18     private Artikel artikel2;
19     private Artikel artikel3;
20
21     /**
22      * Konstruktor fuer die Test-Klasse BestandsverwalterTest
23      */
24     public BestandsverwalterTest()
25     {
26     }
27
28     /**
29      * Setzt das Testgeruest fuer den Test.
30      *
31      * Wird vor jeder Testfall-Methode aufgerufen.
32      */
33     @Before
34     public void setUp()
35     {
36         bestands1 = new Bestandsverwalter();
37         artikel1 = new Artikel(1, "Fahrrad");
38         artikel2 = new Artikel(2, "Zelt");
39         artikel3 = new Artikel(10, "Auto");
40         bestands1.neuerArtikel(artikel1);
41         bestands1.neuerArtikel(artikel2);
42         bestands1.neuerArtikel(artikel3);
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     }
54 }
```

```
55 |     @Test
56 |     public void findeArtikel()
57 |     {
58 |         assertEquals(artikel1, bestands1.findeArtikel(1));
59 |     }
60 |
61 |
62 | }
```

```
1 import java.util.ArrayList;
2
3 /**
4  * Verwalte den Bestand eines Unternehmens.
5  * Der Bestand ist beschrieben durch einen oder
6  * mehrere Artikel.
7 *
8  * @author (Ihr Name)
9  * @version (eine Versionsnummer oder ein Datum)
10 */
11 public class Bestandsverwalter
12 {
13     // Das Lager mit den Artikeln
14     private ArrayList<Artikel> lager;
15
16     /**
17      * Initialisiere den Bestandsverwalter.
18      */
19     public Bestandsverwalter()
20     {
21         lager = new ArrayList<Artikel>();
22     }
23
24     /**
25      * Führe einen neuen Artikel im Lager ein.
26      * @param artikel der Artikel, der neue eingeführt werden soll
27      */
28     public void neuerArtikel(Artikel artikel)
29     {
30         if(findeArtikel(artikel.gibNummer()) == null)
31         {
32             lager.add(artikel);
33         }
34     }
35
36     /**
37      * Nimm eine Lieferung eines Artikels in das Lager auf.
38      * Erhöhe den Lagerbestand um die angegebene Menge.
39      * @param nummer die Artikelnummer des Artikels
40      * @param menge die angelieferte Menge
41      */
42     public void aufnehmen(int nummer, int menge)
43     {
44         Artikel artikel = findeArtikel(nummer);
45         if(!(artikel == null))
46         {
47             artikel.erhoeheBestand(menge);
48         }
49     }
50
51     /**
52      * Versuche einen Artikel mit der angegebenen Nummer im
53      * Bestand zu finden.
54      * @param nummer die Nummer des zu findenden Artikels.
```

```
55 * @return den gefundenen Artikel oder null, falls kein
56 * passender Artikel gefunden wird.
57 */
58 public Artikel findeArtikel(int nummer)
59 {
60     for(Artikel artikel : lager)
61     {
62         if(artikel.gibNummer() == nummer)
63         {
64             return artikel;
65         }
66     }
67     return null;
68 }

69

70 /**
71 * Finde einen Artikel mit der angegebenen Nummer und
72 * liefere seine aktuelle Menge im Bestand.
73 * Wenn die Nummer auf keinen Artikel passt, wird
74 * Null zurückgeliefert.
75 * @param nummer die Nummer des Artikels
76 * @return die Menge des Artikels im Bestand
77 */
78 public int mengeImBestand(int nummer)
79 {
80     Artikel artikel = findeArtikel(nummer);
81     if(artikel == null)
82     {
83         return 0;
84     }
85     else
86     {
87         return artikel.gibBestand();
88     }
89 }

90

91 /**
92 * Informationen über alle Artikel ausgeben.
93 */
94 public void alleArtikelAnzeigen()
95 {
96     for(Artikel artikel : lager)
97     {
98         System.out.println(artikel.toString());
99     }
100 }

101

102 /**
103 * Geringer Bestand von Artikel ausgeben.
104 */
105 public void geringeArtikelAnzeigen(int level)
106 {
107     for(Artikel artikel : lager)
108     {
```

```
109 |     if(artikel.gibBestand() < level)
110 |     {
111 |         System.out.println(artikel.toString());
112 |     }
113 |
114 | }
115 |
116 | }
```

```
1 /**
2  * Modelliert Informationen zu einem Artikel, der von
3  * einer Firma verkauft wird.
4  *
5  * @author David J. Barnes und Michael Kölling
6  * @version 2016.02.29
7  */
8 public class Artikel
9 {
10     // Die Nummer dieses Artikels
11     private int nummer;
12     // Der Name dieses Artikels
13     private String name;
14     // Der aktuelle Bestand dieses Artikels im Lager
15     private int bestand;
16
17     /**
18      * Konstruktor für Objekte der Klasse Artikel.
19      * Der Anfangsbestand ist Null.
20      * @param nummer die Artikelnummer des Artikels
21      * @param name    der Name des Artikels
22      */
23     public Artikel(int nummer, String name)
24     {
25         this.nummer = nummer;
26         this.name = name;
27         bestand = 0;
28     }
29
30     /**
31      * @return die Artikelnummer
32      */
33     public int gibNummer()
34     {
35         return nummer;
36     }
37
38     /**
39      * @return den Artikelnamen
40      */
41     public String gibName()
42     {
43         return name;
44     }
45
46     /**
47      * @return den Bestand im Lager
48      */
49     public int gibBestand()
50     {
51         return bestand;
52     }
53
54     /**
```

```
55 * @return die Nummer, den Namen und den Lagerbestand als String
56 */
57 public String toString()
58 {
59     return nummer + ":" +
60         name +
61         " Lagerbestand: " + bestand;
62 }
63
64 /**
65 * Erhöhe den Lagerbestand dieses Artikels um den
66 * angegebenen Wert.
67 * @param anzahl die Anzahl der zusätzlichen Artikel
68 *                 dieser Wert muss größer als Null sein
69 */
70 public void erhoeheBestand(int anzahl)
71 {
72     if(anzahl > 0) {
73         bestand += anzahl;
74     }
75     else {
76         System.out.println("Versuchte Aufstockung von " +
77                             name +
78                             " um einen negativen Wert: " +
79                             anzahl);
80     }
81 }
82
83 /**
84 * Verkaufe einen dieser Artikel.
85 * Ein Fehler wird gemeldet, wenn der Artikel nicht auf
86 * Lager ist.
87 */
88 public void verkaufeEinen()
89 {
90     if(bestand > 0) {
91         bestand--;
92     }
93     else {
94         System.out.println(
95             "Versuchter Verkauf eines nicht vorrätigen Artikels: " + name);
96     }
97 }
98 }
99 }
```

```
1 /**
2  * Demonstriert die Klassen Bestandsverwalter und Artikel.
3  * Die Demonstration wird funktionstüchtig, indem die
4  * Klasse Bestandsverwalter vervollständigt wird.
5  *
6  * @author David J. Barnes und Michael Kölling
7  * @version 2016.02.29
8 */
9 public class BestandDemo
10 {
11     // Der Bestandsverwalter
12     private Bestandsverwalter verwalter;
13
14 /**
15  * Erzeuge einen Bestandsverwalter und füttere ihn mit
16  * ein paar Beispielartikeln.
17 */
18 public BestandDemo()
19 {
20     verwalter = new Bestandsverwalter();
21     verwalter.neuerArtikel(new Artikel(132, "Uhrenradio"));
22     verwalter.neuerArtikel(new Artikel(37, "Mobiltelefon"));
23     verwalter.neuerArtikel(new Artikel(23, "Mikrowellenherd"));
24 }
25
26 /**
27  * Eine sehr einfache Demonstration, wie ein Bestandsverwalter
28  * benutzt werden könnte. Informationen zu allen Artikeln werden
29  * angezeigt, die Bestandsmenge eines Artikels wird aufgestockt
30  * und anschließend werden erneut alle Artikel angezeigt.
31 */
32 public void demo()
33 {
34     // Zeige Infos zu allen Artikeln.
35     verwalter.alleArtikelAnzeigen();
36     // Eine Lieferung von 5 Artikeln aufnehmen.
37     verwalter.aufnehmen(132, 5);
38     verwalter.alleArtikelAnzeigen();
39 }
40
41 /**
42  * Zeige Details eines bestimmten Artikels.
43  * Wenn vorhanden, werden Name und Bestandsmenge
44  * angezeigt.
45  * @param nummer die Nummer des zu suchenden Artikels
46 */
47 public void zeigeDetails(int nummer)
48 {
49     Artikel artikel = gibArtikel(nummer);
50     if(artikel != null) {
51         System.out.println(artikel.toString());
52     }
53 }
```

```
55  /**
56   * Verkaufe ein Exemplar eines bestimmten Artikels.
57   * Zeige den Status des Artikels vor und nach dem
58   * Verkauf.
59   * @param nummer die Nummer des verkauften Artikels
60   */
61 public void verkaufeArtikel(int nummer)
62 {
63     Artikel artikel = gibArtikel(nummer);
64
65     if(artikel != null) {
66         zeigeDetails(nummer);
67         artikel.verkaufeEinen();
68         zeigeDetails(nummer);
69     }
70 }
71
72 /**
73  * Hole den Artikel mit der angegebenen Nummer vom
74  * Verwalter.
75  * Eine Fehlermeldung wird ausgegeben, wenn kein
76  * Artikel mit dieser Nummer existiert.
77  * @param nummer die Nummer des Artikels
78  * @return      den Artikel oder null, wenn nicht gefunden wird
79  */
80 public Artikel gibArtikel(int nummer)
81 {
82     Artikel artikel = verwalter.findestArtikel(nummer);
83     if(artikel == null) {
84         System.out.println("Artikel mit Nr: " + nummer +
85                            " ist nicht bekannt.");
86     }
87     return artikel;
88 }
89
90 /**
91  * @return  den Bestandsverwalter
92  */
93 public Bestandsverwalter gibVerwalter()
94 {
95     return verwalter;
96 }
97 }
98 }
```

1 |