



```
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 MusikSammlungTest.
10 *
11 * @author (Ihr Name)
12 * @version (eine Versionsnummer oder ein Datum)
13 */
14 public class MusikSammlungTest
15 {
16     /**
17      * Konstruktor fuer die Test-Klasse MusikSammlungTest
18      */
19     public MusikSammlungTest()
20     {
21     }
22
23     /**
24      * Setzt das Testgeruest fuer den Test.
25      *
26      * Wird vor jeder Testfall-Methode aufgerufen.
27      */
28     @Before
29     public void setUp()
30     {
31     }
32
33     /**
34      * Gibt das Testgeruest wieder frei.
35      *
36      * Wird nach jeder Testfall-Methode aufgerufen.
37      */
38     @After
39     public void tearDown()
40     {
41     }
42
43
44     @Test
45     public void testAlleTracksLoeschen()
46     {
47         MusikSammlung musikSam1 = new MusikSammlung();
48         musikSam1.alleTracksAusgeben();
49         assertEquals(4, musikSam1.gibAnzahlTracks());
50         musikSam1.alleTracksAusgeben();
51         musikSam1.alleTracksLoeschen("BlindLemonJefferson");
52         assertEquals(2, musikSam1.gibAnzahlTracks());
53         musikSam1.alleTracksAusgeben();
54     }
```

55 | }
56 |
57 |
58 |

```
1 import java.io.File;
2 import java.io.FilenameFilter;
3 import java.io.IOException;
4 import java.util.ArrayList;
5
6 /**
7  * Eine Hilfsklasse für unsere Musik-Software. Diese Klasse kann Dateien mit einer
8 bestimmten
9  * Endung aus einem beliebigen Ordner des Dateisystems lesen. Sie geht davon aus,
10 dass der Dateiname
11  * den Interpreten und den Titel enthält.
12 *
13  * Die Dateinamen der Audiotracks müssen daher einem vorgegebenen Standardformat
14 folgen: auf den Namen
15  * des Interpreten folgt, getrennt durch einen Bindestrich der Track-Titel. Zum
16 Beispiel: TheBeatles-HereComesTheSun.mp3
17 *
18 * @author David J. Barnes und Michael Kölling
19 * @version 2011.03.27
20 */
21 public class TrackReader
22 {
23     /**
24      * Erzeuge einen Track-Reader, mit dem Tracks aus dem Musikbibliotheks-Ordner
25 gelesen werden können.
26     */
27     public TrackReader()
28     {
29         // Hier gibt es nichts zu tun
30     }
31
32     /**
33      * Lies Musikdateien mit der angegebenen Endung aus dem angegebenen
34 Bibliotheksordner.
35      * @param ordner der Ordner, in dem nach Dateien gesucht werden soll.
36      * @param suffix die Endung für den Audiotyp.
37     */
38     public ArrayList<Track> liesTracks(String ordner, final String suffix)
39     {
40         File audioOrdner = new File(ordner);
41         ArrayList<Track> tracks = new ArrayList<Track>();
42         File[] audioDateien = audioOrdner.listFiles(new FilenameFilter() {
43             /**
44              * Akzeptiere nur Dateien mit passender Endung.
45              * @param verz das Verzeichnis, in dem die Datei liegt.
46              * @param name der Name der Datei.
47              * @return true, wenn der Namen die gewünschte Endung hat.
48             */
49             public boolean accept(File verz, String name)
50             {
51                 return name.toLowerCase().endsWith(suffix);
52             }
53         });
54         // Füge alle übereinstimmenden Dateien in die Track-Sammlung ein.
55     }
56 }
```

```
50     for(File file : audioDateien) {
51         Track trackDetails = dekodierteDetails(file);
52         tracks.add(trackDetails);
53     }
54     return tracks;
55 }
56
57 /**
58 * Versuche die Angaben zu Interpret und Titel aus dem Dateinamen zu
59 * extrahieren.
60 * Geht davon aus, dass der Dateiname folgendes Format hat:
61 *   interpret-titel.mp3
62 * @param datei die Track-Datei.
63 * @return ein Track-Objekt mit den entsprechenden Angaben.
64 */
65 private Track dekodierteDetails(File datei)
66 {
67     // Die benötigte Information.
68     String interpret = "unbekannt";
69     String titel = "unbekannt";
70     String dateiname = datei.getPath();
71
72     // Sucht im Dateinamen nach dem Interpreten und dem Titel.
73     String details = datei.getName();
74     String[] komponenten = details.split("-");
75
76     if(komponenten.length == 2) {
77         interpret = komponenten[0];
78         String titelKomponente = komponenten[1];
79         // Dateierweiterung entfernen.
80         komponenten = titelKomponente.split("\\.");
81         if(komponenten.length >= 1) {
82             titel = komponenten[0];
83         }
84         else {
85             titel = titelKomponente;
86         }
87     }
88     return new Track(interpret, titel, dateiname);
89 }
90 }
```

```
1 import java.io.BufferedInputStream;
2 import java.io.FileInputStream;
3 import java.io.InputStream;
4 import java.io.IOException;
5 import javazoom.jl.decoder.JavaLayerException;
6 import javazoom.jl.player.AudioDevice;
7 import javazoom.jl.player.FactoryRegistry;
8 import javazoom.jl.player.advanced.AdvancedPlayer;
9
10 /**
11 * Stellt mithilfe der javazoom-Bibliothek die grundlegende Funktionalität zum
12 * Abspielen von MP3-Dateien zur Verfügung.
13 * siehe http://www.javazoom.net/
14 *
15 * @author David J. Barnes und Michael Kölling
16 * @version 31.07.2011
17 */
18 public class MusikPlayer
19 {
20     // Das aktuell verwendete Abspielgerät. Kann null sein.
21     private AdvancedPlayer player;
22
23     /**
24      * Konstruktor für Objekte der Klasse MusicPlayer
25      */
26     public MusikPlayer()
27     {
28         player = null;
29     }
30
31     /**
32      * Spielt einen Teil der gegebenen Datei ab.
33      * Die Methode kehrt zurück, sobald sie mit Abspielen fertig ist.
34      * @param dateiname die abzuspielende Datei.
35      */
36     public void dateiAnspielen(String dateiname)
37     {
38         try {
39             playerVorbereiten(dateiname);
40             player.play(500);
41         }
42         catch(JavaLayerException e) {
43             meldeProblem(dateiname);
44         }
45         finally {
46             killPlayer();
47         }
48     }
49
50     /**
51      * Spielt die gegebene Audiodatei ab.
52      * Die Methode kehrt zurück, sobald der Abspielvorgang gestartet wurde.
53      * @param dateiname die abzuspielende Datei.
54      */
```

```
55  public void starteAbspielen(final String dateiname)
56  {
57      try {
58          playerVorbereiten(dateiname);
59          Thread playerThread = new Thread() {
60              public void run()
61              {
62                  try {
63                      player.play(5000);
64                  }
65                  catch(JavaLayerException e) {
66                      meldeProblem(dateiname);
67                  }
68                  finally {
69                      killPlayer();
70                  }
71              }
72          };
73          playerThread.start();
74      }
75      catch (Exception ex) {
76          meldeProblem(dateiname);
77      }
78  }
79
80  public void stop()
81  {
82      killPlayer();
83  }
84
85 /**
86 * Bereite den Player für das Abspielen der gegebenen Datei vor.
87 * @param dateiname der Name der abzuspielenden Datei.
88 */
89 private void playerVorbereiten(String dateiname)
90 {
91     try {
92         InputStream is = gibEingabestream(dateiname);
93         player = new AdvancedPlayer(is, erzeugeAudioderaet());
94     }
95     catch (IOException e) {
96         meldeProblem(dateiname);
97         killPlayer();
98     }
99     catch(JavaLayerException e) {
100         meldeProblem(dateiname);
101         killPlayer();
102     }
103 }
104
105 /**
106 * Liefere einen Eingabestream für die gegebene Datei.
107 * @param dateiname die zu öffnende Datei.
108 * @throws IOException wenn die Datei nicht geöffnet werden kann.
```

```
109     * @return ein InputStream für die Datei.  
110    */  
111    private InputStream gibEingabestream(String dateiname)  
112        throws IOException  
113    {  
114        return new BufferedInputStream(  
115            new FileInputStream(dateiname));  
116    }  
117  
118    /**  
119     * Erzeuge ein Audiogerät.  
120     * @throws JavaLayerException wenn das Gerät nicht erzeugt werden kann.  
121     * @return ein Audiogerät.  
122     */  
123    private AudioDevice erzeugeAudigeraet()  
124        throws JavaLayerException  
125    {  
126        return FactoryRegistry.systemRegistry().createAudioDevice();  
127    }  
128  
129    /**  
130     * Beende den Player, sofern ein solcher existiert.  
131     */  
132    private void killPlayer()  
133    {  
134        synchronized(this) {  
135            if(player != null) {  
136                player.stop();  
137                player = null;  
138            }  
139        }  
140    }  
141  
142    /**  
143     * Berichte über ein Problem beim Abspielen der gegebenen Datei.  
144     * @param dateiname die abgespielte Datei.  
145     */  
146    private void meldeProblem(String dateiname)  
147    {  
148        System.out.println("Es gab ein Problem beim Abspielen von: " + dateiname);  
149    }  
150  
151 }  
152 }
```

```
1 import java.util.ArrayList;
2 import java.util.Iterator;
3 import java.util.Random;
4
5 /**
6  * Eine Klasse zur Verwaltung von Audiotracks.
7  * Die einzelnen Tracks können abgespielt werden.
8  *
9  * @author David J. Barnes und Michael Kölling.
10 * @version 31.07.2011
11 */
12 public class MusikSammlung
13 {
14     // Eine ArrayList, in der die Musik-Tracks gespeichert werden können.
15     private ArrayList<Track> tracks;
16     // Ein Player zum Abspielen der Musik-Tracks.
17     private MusikPlayer player;
18     // Ein Reader, der Musikdateien lesen und als Tracks laden kann
19     private TrackReader reader;
20
21     private Random random;
22
23     /**
24      * Erzeuge eine MusikSammlung
25      */
26     public MusikSammlung()
27     {
28         tracks = new ArrayList<Track>();
29         player = new MusikPlayer();
30         reader = new TrackReader();
31         random = new Random();
32         liesBibliothek("audio");
33         System.out.println("Musikbibliothek wurde geladen. " + gibAnzahlTracks()
34 + " Tracks.");
35         System.out.println();
36     }
37
38     /**
39      * Füge der Sammlung eine Track-Datei hinzu.
40      * @param dateiname der Dateiname des hinzuzufügenden Tracks.
41      */
42     public void dateiHinzufuegen(String dateiname)
43     {
44         tracks.add(new Track(dateiname));
45     }
46
47     /**
48      * Füge der Sammlung einen Track hinzu.
49      * @param dateiname der hinzuzufügende Track.
50      */
51     public void trackHinzufuegen(Track track)
52     {
53         tracks.add(track);
54     }
```

```
54
55 /**
56 * Spielt einen Track aus der Sammlung.
57 * @param index der Index des abzuspielenden Tracks.
58 */
59 public void spieleTrack(int index)
60 {
61     if(gueltigerIndex(index)) {
62         Track track = tracks.get(index);
63         player.starteAbspielen(track.gibDateiname());
64         System.out.println("Sie hoeren gerade: " + track.gibInterpret() + " - "
65 + track.gibTitel());
66     }
67 }
68 /**
69 * Liefere die Anzahl der Tracks in dieser Sammlung.
70 * @return die Anzahl der tracks in dieser Sammlung.
71 */
72 public int gibAnzahlTracks()
73 {
74     return tracks.size();
75 }
76
77
78 /**
79 * Gib einen Track aus der Sammlung auf die Konsole aus.
80 * @param index der Index des auszugebenden Tracks.
81 */
82 public void trackAusgeben(int index)
83 {
84     System.out.print("Track " + index + ": ");
85     Track track = tracks.get(index);
86     System.out.println(track.gibDetails());
87 }
88
89 /**
90 * Gib eine Liste aller Tracks in der Sammlung aus.
91 */
92 public void alleTracksAusgeben()
93 {
94     System.out.println("Track-Liste: ");
95
96     for(Track track : tracks) {
97         System.out.println(track.gibDetails());
98     }
99     System.out.println();
100 }
101
102 /**
103 * Liste alle Tracks zu einem gegebenen Interpreten.
104 * @param interpret der Name des Interpreten.
105 */
106 public void bestimmteTracksAusgeben(String interpret)
```

```
107  {
108      for(Track track : tracks) {
109          if(track.gibInterpret().contains(interpret)) {
110              System.out.println(track.gibDetails());
111          }
112      }
113  }
114
115
116 /**
117 * Entferne einen Track aus der Sammlung.
118 * @param index der Index, des zu entfernenden Tracks.
119 */
120 public void entferneTrack(int index)
121 {
122     if(gueltigerIndex(index)) {
123         tracks.remove(index);
124     }
125 }
126
127 /**
128 * Spielle den ersten Track aus der Sammlung, falls vorhanden.
129 */
130 public void spieleErsten()
131 {
132     if(tracks.size() > 0) {
133         player.starteAbspielen(tracks.get(0).gibName());
134     }
135 }
136
137 /**
138 * Stoppt den Player.
139 */
140 public void beendeAbspielen()
141 {
142     player.stop();
143 }
144
145 /**
146 * Stelle fest, ob der gegebene Index für die Sammlung gültig ist.
147 * Falls nicht, wird eine Fehlermeldung ausgegeben.
148 * @param index der zu prüfende Index.
149 * @return true, wenn der Index gültig ist, andernfalls false.
150 */
151 private boolean gueltigerIndex(int index)
152 {
153     // Der Rückgabewert.
154     // Setze den Rückgabewert abhängig davon, ob der Index gültig ist oder
155     // nicht.
156     boolean gueltig;
157
158     if(index < 0) {
159         System.out.println("Indizes können nicht negativ sein: " + index);
160         gueltig = false;
161     }
162 }
```

```
161     else if(index >= tracks.size()) {
162         System.out.println("Index ist zu gross: " + index);
163         gueltig = false;
164     }
165     else {
166         gueltig = true;
167     }
168     return gueltig;
169 }
170
171 private void liesBibliothek(String ordnerName)
172 {
173     ArrayList<Track> tempTracks = reader.liesTracks(ordnerName, ".mp3");
174
175     // Alle Tracks in die Sammlung einfügen.
176     for(Track track : tempTracks) {
177         trackHinzufuegen(track);
178     }
179 }
180
181 public void loescheAlleTracks(String interpret)
182 {
183     for (Track track: tracks)
184     {
185         if(track.gibInterpret().equals(interpret))
186         {
187             tracks.remove(track);
188         }
189     }
190 }
191
192 public void alleTracksLoeschen(String interpret)
193 {
194     Iterator<Track> it = tracks.iterator();
195     while(it.hasNext())
196     {
197         Track track = it.next();
198         if(track.gibInterpret().equals(interpret))
199         {
200             it.remove();
201         }
202     }
203 }
204
205 public void zufaelligenTrackAbspielen(Random random)
206 {
207 }
208 }
209 }
```

```
1 /**
2  * Speichere die Details von Musiktiteln (Tracks),
3  * wie z.B. den Interpreten, den Titel und den Dateinamen.
4  *
5  * @author David J. Barnes und Michael Kölling
6  * @version 31.07.2011
7  */
8 public class Track
9 {
10     // Der Interpret.
11     private String interpret;
12     // Der Titel des Tracks.
13     private String titel;
14     // Wo der Track gespeichert ist.
15     private String dateiname;
16
17     /**
18      * Konstruktor für Objekte der Klasse Track.
19      * @param interpret der Interpret des Titels.
20      * @param titel der Titel des Tracks.
21      * @param dateiname die Track-Datei.
22      */
23     public Track(String interpret, String titel, String dateiname)
24     {
25         setzeDetails(interpret, titel, dateiname);
26     }
27
28     /**
29      * Konstruktor für Objekte der Klasse Track.
30      * Geht davon aus, dass der Dateiname nicht dekodiert werden kann, um
31      * den tatsächlichen Interpreten und Titel zu ermitteln.
32      * @param dateiname die Track-Datei.
33      */
34     public Track(String dateiname)
35     {
36         setzeDetails("unbekannt", "unbekannt", dateiname);
37     }
38
39     /**
40      * Liefere den Interpreten.
41      * @return der Interpret.
42      */
43     public String gibInterpret()
44     {
45         return interpret;
46     }
47
48     /**
49      * Liefere den Titel.
50      * @return der Titel.
51      */
52     public String gibTitel()
53     {
54         return titel;
```

```
55    }
56
57    /**
58     * Liefere den Dateinamen.
59     * @return der Dateiname.
60     */
61    public String gibDateiname()
62    {
63        return dateiname;
64    }
65
66    /**
67     * Liefer Details über den Track: Interpret, Titel und Dateiname.
68     * @return die Track-Details.
69     */
70    public String gibDetails()
71    {
72        return interpret + ":" + titel + " (Datei: " + dateiname + ")";
73    }
74
75    /**
76     * Lege die Details des Tracks fest.
77     * @param interpret der Interpret des Titels.
78     * @param titel der Titel des Tracks.
79     * @param dateiname die Track-Datei.
80     */
81    private void setzeDetails(String interpret, String titel, String dateiname)
82    {
83        this.interpret = interpret;
84        this.titel = titel;
85        this.dateiname = dateiname;
86    }
87
88}
```

1 Projekt: Musiksammlung-v5. Ein Projekt zum Sammeln von Audiodateien.
2 Autoren: David Barnes und Michael Kölling
3
4 Dieses Projekt ist Teil des Zusatzmaterials zum Buch
5
6 Java lernen mit BlueJ - eine Einführung in die
7 objektorientierte Programmierung, 5. Auflage
8 David J. Barnes und Michael Kölling
9 Pearson Education Deutschland, 2012
10
11 Startpunkt: Erzeugen Sie ein Builder-Objekt und rufen Sie dann dessen
12 getCollection-Methode auf. Nutzen Sie den Get-Schalter des Ergebnisfenster,
13 um das Musiksammlung-Objekt auf die Objektbench zu setzen.
14
15