Musterbeispiel für den 2.Test aus Programmierpraxis

Implementieren Sie alle Aufgaben in den dafür vorgesehenen Dateien im Ordner "impl".

- Bitte lesen Sie die Angabe **komplett** durch, bevor Sie mit der Umsetzung beginnen.
- Halten Sie sich genau an die Angabe und geben Sie nur die geforderten Daten aus.
- Haben Sie **Schwierigkeiten** bei der Umsetzung einer der Teilaufgaben, **versuchen Sie andere Teile zuerst umzusetzen** und wenden Sie sich zuletzt nochmal der ungelösten Teilaufgabe zu. Sie erhalten auch Punkte für nicht vollständig implementierte Teilaufgaben

Die `JukeBox` verwaltet eine Musik-Sammlung. Eine Musik-Sammlung wird repräsentiert durch Bibliotheken, welche mehrere Playlists umfassen, die jeweils durch einen Namen gekennzeichnet sind. Jede Playlist ist eine sortierte Sammlung von Songs.

Aufgabenstellung

Die Aufgabe umfasst die **Implementierung der geforderten Methoden in den angegeben Klassen (20 Punkte)** und die **Beantwortung der Zusatzfragen (5 Punkte)**. Als freiwillige Zusatzaufgabe können Sie einen zusätzlichen Testfall erstellen: Für einen sinnvollen und gut kommentierten Testfall können Sie einen zusätzlichen Punkt bekommen, der einen an einer anderen Stelle verlorenen Punkt ausgleicht (insgesamt können Sie jedoch nicht mehr als 25 Punkte erreichen).

Funktionale Anforderungen

Sie entwickeln ein Musikverwaltungsprogramm. Die Musik wird hierbei durch Lieder (Song) repräsentiert; **jeder Song** besitzt einen **Titel**, einen **Interpreten** und eine **Laufzeit in Sekunden**. Ein Song ist in seinen Eigenschaften **nicht veränderlich**.

Mehrere Songs können zu einer **Playlist** (`Playlist`) zusammengefasst werden. Eine Playlist hat eine bestimmte **Reihenfolge**, ein Song darf jedoch nur einmal in der Playlist enthalten sein. Eine Playlist soll einfach kopiert werden können. Weiters soll eine Playlist auch als Ergebnis des Aneinanderfügens zweier vorhandener Playlists erstellbar sein.

Aus einer Playlist kann außerdem eine neue Playlist erstellt werden, in der alle Songs eines bestimmten Künstlers aus der alten Playlist vorhanden sind. Darüber hinaus soll es auch die Möglichkeit geben, die Anzahl der Songs eines bestimmten Künstlers innerhalb der Playlist zu zählen.

Die **Playlists** werden **zentral in einer Bibliothek** (`Library`) gespeichert. Dabei soll ein Import aller Playlists in eine neue Library möglich sein, wobei die Playlists tatsächlich kopiert werden, das heißt, eine Änderung der Playlist in der neuen Library beeinflusst nicht die originale Library.

Nicht-funktionale Anforderungen

Nutzen Sie wenn möglich immer bestehenden Code und **vermeiden Sie so doppelte Codestücke**. Sie können bei Bedarf auch zusätzliche Methoden einführen. Dies sollten Sie jedoch nur dann machen, wenn es notwendig ist oder im Rahmen von jedoch nur dann

machen, wenn es notwendig ist oder im Rahmen von **Codewiederverwendung** Sinn macht. Achten Sie bei Ihren Klassen auf **korrekte Datenkapselung und Datenabstraktion**, also insbesondere auf die richtigen (Sichtbarkeits-)Modifikatoren bei Methoden und Datenfeldern.

Vorgehensweise

Implementierungsreihenfolge

Folgende Vorgehensweise ist bei der Umsetzung der Aufgabe empfehlenswert:

- Beginnen Sie mit der Implementierung der Klasse Song. Hierbei sind besonders der Konstruktor mit drei Parametern sowie die `getTitle`, `getArtist`, `getDuration` und die `toString`-Methode relevant. Nutzen Sie den vorgefertigten Testfall 1 um diese grundlegende Funktionalität zu testen.

- Setzen Sie im nächsten Schritt mit der Implementierung der Klasse `Playlist` fort. Hierbei sollten Sie wiederum mit dem Konstruktor und der `toString`-Methode beginnen. Weiters sollten Sie die `addSong`-Methode implementieren. Der Testfall 2 überprüft genau diese Funktionalität.

Haben Sie diese beiden Schritte umgesetzt, können Sie nun die weiteren Teilaufgaben umsetzen.

Testen

Kompilieren Sie früh und oft und versuchen Sie Fehler sofort zu beheben. **Testen Sie nach jedem Kompilieren** Ihre Implementierung. Es ist im Rahmen des Tests nicht erforderlich selbst Eingaben vom User zu verarbeiten. Erzeugen Sie statt dessen zu Testzwecken in der Methode `testing` Objekte der von Ihnen implementierten Klassen und rufen Sie die verschiedenen Methoden auf. Zur Unterstützung finden Sie zu jeder Teilaufgabe bereits **fertige Testfälle** in der Methode `testCases` der Klasse `Jukebox`.

Um die einzelnen Testfälle aufzurufen, nutzen Sie

- **`java Jukebox`** um die testing-Methode aufzurufen
- **`java Jukebox`** ***`[number]`*** um die `testCases`-Methode und somit den

entsprechenden Testfall aufzurufen

Die geforderten Ausgaben finden Sie im Abschnitt Testfälle und zusätzlich auch in der Beschreibung der jeweiligen Testfälle in der Klasse `Jukebox`.

Klassen und Methoden

Die folgenden Klassen und Methoden sind zu implementieren. Falls nicht anders angegeben, können Sie davon ausgehen, dass die den Methoden bei einem Aufruf übergebenen Werte

gültig sind (daher beispielsweise keine `IndexOutOfBoundsException` verursachen). Sie müssen die Gültigkeit daher nicht überprüfen.

`Jukebox`

Diese Klasse ist ausführbar und beinhaltet daher die `main`-Methode.

`public static void testing()`

Testen Sie in dieser Methode die Implementierung Ihres Programmes durch Objekt-Instanzierungen und Methodenaufrufe. Geben Sie Ausgaben (Rückgaben von Methoden, etc.) auf `System.out` aus.

Erstellen Sie zuletzt einen sinnvollen Testfall und beschreiben Sie kurz in einem Kommentar, wieso Sie diesen Testfall gewählt haben und was Sie damit überprüfen. Für einen sinnvollen und gut kommentierten Testfall können Sie einen zusätzlichen Punkt bekommen, der einen an einer anderen Stelle verlorenen Punkt ausgleicht (insgesamt können Sie jedoch nicht mehr als 30 Punkte erreichen).
(1 Zusatzpunkt)

Die weiteren Methoden in der Klasse `Jukebox` **sollen nicht verändert werden**. Diese Methoden dienen dem Ausführen der vorgefertigten Testfälle und werden nicht beurteilt. Nutzen Sie diese Testfälle um Fehler in Ihrem Programm zu entdecken und auch als Anregungen für eigene Testfälle. Weitere Informationen zum Testen finden Sie unter dem Punkt Vorgehensweise.

`Song`

Diese Klasse repräsentiert ein Lied aus der Musiksammlung. Ein Song ist der kleinste Bestandteil einer Musiksammlung und ist charakterisiert durch einen Titel, einen Interpreten und die Laufzeit in (ganzen) Sekunden.

`Playlist`

Diese Klasse repräsentiert eine Playlist aus der Musiksammlung. Eine Playlist ist eine Ansammlung von Songs, die in einer bestimmten Reihenfolge (jeder Song jedoch nur einmal) vorkommen dürfen.

`Library`

Diese Klasse repräsentiert eine Musikbibliothek, also eine Ansammlung an Playlists. Jede Playlist wird hierbei durch einen Namen identifiziert.

Zusatzfragen

Halten Sie Ihre Antworten kurz und beschränken Sie sich auf das Wesentliche.

Frage 1: Collections

- Welche Einschränkung haben Collections gegenüber Arrays (insbesondere in Hinblick auf die verwendbaren Typen)? Beschreiben Sie in diesem Zusammenhang auch das Konzept und den Vorteil des Auto-Boxing. Gäbe es Autoboxing nicht, wie müssten Sie folgenden Code zum Einfügen in eine `ArrayList` umbauen?

  ```
  arrayList.add(5);
  ```

- Nennen Sie zwei unterschiedliche Möglichkeiten, um alle Elemente einer Collection zu durchlaufen und geben Sie jeweils ein kurzes Codebeispiel an. Beschreiben Sie kurz Vor- und Nachteile der beiden Möglichkeiten.

Frage 2: Unveränderbare Objekte

- Betrachten Sie folgende Klasse:

  ```
  public class Vector {

       private double x;
       private double y;

       public Vector (double x, double y) {
            this.x = x;
            this.y = y;
       }

       public void add(Vector v) {
            this.x += v.x;
            this.y += v.y;
       }

       public String toString() {
            return "(" + this.x + "," + this.y + ")";
       }

  }
  ```

  Ändern Sie die Klasse `Vector` so, dass Instanzen der Klasse unveränderbar sind. Die vorhandene Funktionalität soll dabei erhalten bleiben, daher soll es nach wie vor möglich sein, eine Addition von zwei `Vector`-Objekten durchzuführen. Beschreiben Sie die notwendigen Änderungen in dieser Klasse (Methodensignaturen, Datenelemente, ...) und erklären Sie, warum diese erforderlich sind.

- Im folgenden Ausschnitt eines Programms wird `Vector` verwendet:

  ```
  Vector v = new Vector(3,4);
  v.add(new Vector(2,2));
  System.out.println(v);
  ```

Welche Veränderungen müssen vorgenommen werden, sodass `v` nach den zuvor beschriebenen Änderungen in der Klasse `Vector` weiterhin das Ergebnis der Addition enthält und dieses korrekt ausgegeben wird?

Testfälle

Beachten Sie die oben beschriebene bezüglich Testen empfohlene Vorgehensweise!

Testfall 1	INPUT	`Song song1 = new Song("Atreyu", "Falling Down", 215);`
		`System.out.println(song1.getArtist());`
		`System.out.println(song1.getTitle());`
		`System.out.println(song1.getDuration());`
		`System.out.println(song1);`

	OUTPUT	`Atreyu`
		`Falling Down`
		`215`
		`Falling Down - Atreyu (215 sec)`

	?	Überprüft die korrekte Implementierung der Song-Klasse

Testfall 2	INPUT	`Song song1 = new Song("Atreyu", "Falling Down", 215);`
		`Song song2 = new Song("Blur", "Song 2", 126);`
		`Song song3 = new Song("Amy MacDonald", "My Only One", 212);`
		`Playlist playlist1 = new Playlist();`
		`playlist1.addSong(song1);`
		`playlist1.addSong(song2);`
		`System.out.println(playlist1);`

	OUTPUT	`true`
		`false`
		`Falling Down - Atreyu (215 sec)`
		`Song 2 - Blur (126 sec)`
		`My Only One - Amy MacDonald (212 sec)`
		`553 sec`

	?	Überprüft die korrekte Basisimplementierung der Playlist-Klasse

<table>
<tr><td rowspan="3">Testfall 3</td><td>INPUT</td><td>

```
Song song1 = new Song("Atreyu", "Falling Down", 215);
Song song2 = new Song("Amy MacDonald", "My Only One", 212);
Song song3 = new Song("Pink Floyd", "Shine On You Crazy
Diamond", 810);
Song song4 = new Song("Jamie Cullum", "Mixtape", 299);
Song song5 = new Song("Annie Stettin", "Beats For You",
245);
Song song6 = new Song("Jamie Cullum", "Back To The Ground",
277);

Playlist playlist1 = new Playlist();
playlist1.addSong(song1);

Playlist playlist2 = new Playlist(playlist1);
playlist2.addSong(song2);

Playlist playlist3 = new Playlist(playlist1, playlist2);
playlist3.addSong(song3);
System.out.println(playlist3);

Playlist playlist4 = new Playlist();

playlist4.addSong(song4);
playlist4.addSong(song5);
playlist4.addSong(song6);

Playlist jamieCullum =
playlist4.extractPlaylistByArtist("Jamie Cullum");
System.out.println(jamieCullum);
System.out.println(playlist4.countSongsByArtist("Jamie
Cullum"));
```

</td></tr>
<tr><td>OUTPUT</td><td>

```
Falling Down - Atreyu (215 sec)
My Only One - Amy MacDonald (212 sec)
Shine On You Crazy Diamond - Pink Floyd (810 sec)
1237 sec
Mixtape - Jamie Cullum (299 sec)
Back To The Ground - Jamie Cullum (277 sec)
576 sec
2
```

</td></tr>
<tr><td>?</td><td>Überprüft die erweiterte Implementierung (Kopie, Zusammenfügung, Extraktion) der <code>Playlist</code>-Klasse</td></tr>
</table>

<table>
<tr><td rowspan="3">Testfall 4</td><td>INPUT</td><td>

```
Song song1 = new Song("Atreyu", "Falling Down", 215);
Song song2 = new Song("Blur", "Song 2", 126);
Song song3 = new Song("Amy MacDonald", "My Only One",
212);
Song song4 = new Song("Pink Floyd", "Shine On You
Crazy Diamond", 810);
Library library = new Library();

Playlist playlist1 = new Playlist();
playlist1.addSong(song1);
playlist1.addSong(song2);

Playlist playlist2 = new Playlist();
playlist2.addSong(song3);
playlist2.addSong(song4);

library.addPlaylist("Favourites", playlist1);

System.out.println(library.getPlaylist("Favourites")
== playlist1);

library.addPlaylist("Favourites", playlist2);
System.out.println(library.getPlaylist("Favourites")
== playlist1);
System.out.println(library.getPlaylist("Favourites")
== playlist2);

library.removePlaylist("Favourites");
System.out.println(library.getPlaylist("Favourites")
== null);
```

</td></tr>
<tr><td>OUTPUT</td><td>

```
true
false
true
true
```

</td></tr>
<tr><td>?</td><td>Überprüft die korrekte Basisimplementierung der Playlist-Klasse</td></tr>
</table>

Testfall 5

INPUT

```
Song song1 = new Song("Atreyu", "Falling Down", 215);
Song song2 = new Song("Blur", "Song 2", 126);
Song song3 = new Song("Amy MacDonald", "My Only One", 212);
Song song4 = new Song("Pink Floyd", "Shine On You Crazy
Diamond", 810);
Song song5 = new Song("Jamie Cullum", "Mixtape", 299);
Song song6 = new Song("Annie Stettin", "Beats For You",
245);

Library library1 = new Library();

Playlist playlist1 = new Playlist();
playlist1.addSong(song1);
playlist1.addSong(song2);

Playlist playlist2 = new Playlist();
playlist2.addSong(song3);
playlist2.addSong(song4);

Playlist playlist3 = new Playlist();
playlist3.addSong(song5);
playlist3.addSong(song6);

library1.addPlaylist("Rock", playlist1);
library1.addPlaylist("Lieblingslieder", playlist2);
library1.addPlaylist("Unterwegs", playlist3);

Library library2 = new Library(library1,
                               new String[]{
                                      "Unterwegs",
                                      "Lieblingslieder"
                               });

System.out.println(library2.getPlaylist("Rock"));
System.out.println(library2.getPlaylist("Lieblingslieder"));
System.out.println(library2.getPlaylist("Unterwegs"));
```

OUTPUT

```
null
My Only One - Amy MacDonald (212 sec)
Shine On You Crazy Diamond - Pink Floyd (810 sec)
1022 sec
Mixtape - Jamie Cullum (299 sec)
Beats For You - Annie Stettin (245 sec)
544 sec
```

?

Überprüft die Implementierung der Import-Funktion der Library-Klasse