hochschule mannheim

FAKULTÄT FÜR INFORMATIK

Prof. Dr. Peter Knauber

PR1 – Programmieren 1 Bachelor-Studiengang Informatik WS2017/2018

Übungsblatt 10: Klassen und Objekte, ADTs, Interfaces

Ausgabe: 19.12.17
Abgabe: 8.1.18 12:00 Uhr elektronisch mittels Git

Aufgabe 1

a) Erstellen Sie eine Klasse `SongImplementation`, um den Titel eines Songs, den Namen des zugehörigen Albums sowie einen oder mehrere zugehörige Musiker/Bands zu speichern. Die Klasse soll einen geeigneten Konstruktor besitzen und die beiden Interfaces `Song` und `Comparable<Song>` implementieren. Das Interface `Song` ist wie folgt definiert:

```
public interface Song {

	/**
	 * Liefert den Namen des Songs
	 */
	String getSongName();

	/**
	 * Liefert den Namen des zum Song gehoerigen Albums.
	 * Die Methode wird nur zu Testzwecken benoetigt!
	 */
	String getAlbumName();

	/**
	 * Liefert ein passend dimensioniertes Array mit einem oder mehreren Kuenstlern/Bands.
	 * Die Methode wird nur zu Testzwecken benoetigt!
	 */
	String[] getArtists();

	/**
	 * Liefert eine gut lesbare Darstellung der Daten des Songs
	 */
	String toString();

}
```

Das Interface `Comparable<Song>` (`Song` dient als Parameter für das Interface `Comparable`) gehört zum Standard-Java-Sprachumfang.
Die Klasse `SongImplementation` wird also wie folgt vereinbart:

```
public class SongImplementation implements Song, Comparable<Song> {
	...
}
```

b) Wir wollen unsere Songs in einer Liste verwalten, in der sie automatisch (Methode `insert`) anhand des Namens alphabetisch aufsteigend sortiert werden. Die Song-Namen sind also lexikographisch geordnet.
Dafür nutzen wir einen abstrakten Datentyp "OrderedList", der in Java durch das folgende Interface beschrieben wird:

```
public interface OrderedList {

        /**
         * Fuegt das Objekt der Klasse Song so in die Liste ein,
         * dass alle Song-Namen in der Liste lexikographisch geordnet sind.
         */
        void insert(Song song);

        /**
         * Liefert die Position der Liste, an der sich der Song mit dem
         * angegebenen Namen befindet; dabei hat die erste Listenposition
         * die Positionsnummer 0.
         * Ist kein Song mit dem angegebenen Namen in der Liste enthalten,
         * dann liefert die Methode -1 als Ergebnis.
         */
        int indexOf(String songName);

        /**
         * Loescht den Song an der angegebenen Position aus der Liste;
         * dabei hat die erste Listenposition die Positionsnummer 0.
         * @throws PRException wenn die Positionsangabe ungueltig ist.
         */
        void delete(int position) throws PRException;

        /**
         * Liefert die Laenge der Liste, d.h. die Anzahl der Listenelemente.
         */
        int size();

        /**
         * Liefert den Song an der angegebenen Position der Liste;
         * dabei hat die erste Listenposition die Positionsnummer 0.
         * @throws PRException wenn die Positionsangabe ungueltig ist.
         */
        Song get(int position) throws PRException;

}
```

Erstellen Sie eine Klasse `ArrayList`, welche das Interface `OrderedList` implementiert, indem ein Array zur Ablage der enthaltenen `Songs` benutzt wird.
Auf `Song`-Objekte dürfen Sie nur mit den Methoden aus dem Interface zugreifen!

c) Erstellen Sie eine Klasse `LinkedList`, welche das Interface `OrderedList` implementiert, indem eine verkette Liste (wie in der Vorlesung besprochen) zur Ablage der enthaltenen `Songs` benutzt wird.
Auf `Song`-Objekte dürfen Sie nur mit den Methoden aus dem Interface zugreifen!

d) Erstellen Sie eine Klasse `TestClass` mit einer `main`-Methode wie üblich, um Ihre Listen-Implementierungen zu testen.

Benutzen Sie folgende Testdaten:
Die Datei "songs.txt" enthält eine Menge von Song-Testdaten, die in jeder Zeile durch Semikolon getrennt jeweils den Titel eines Songs, den Namen des zugehörigen Albums sowie einen oder mehrere Musiker enthalten. Sie können die Datei ganz einfach mit der `MakeItSimple`-Methode `readStringArray` in ein Array von Strings lesen und zum Beispiel mit Ihrer `split`-Methode von Übungsblatt 7 in Teilstrings aufteilen. Aus dem Ergebnis lässt sich dann mittels Konstruktor ein `Song`-Objekt erzeugen und mittels `insert` in eine Liste packen.

e) Diese Teilaufgabe ist einen Bonuspunkt wert:
Implementieren Sie die Methode `contains` aus dem Interface `OrderedList` in Form einer *binären Suche* (nur) in der Klasse `ArrayList`. Dazu gehört:

- Informieren Sie sich, wie eine binäre Suche funktioniert.
- Implementieren Sie eine solche Suche.
- Bereiten Sie sich darauf vor zu erklären, wie Ihre Implementierung funktioniert, was die Voraussetzungen dafür sind und warum eine binäre Suche besser ist als eine lineare Suche. Was müsste man tun, um eine binäre Suche auch für die Klasse `LinkedList` zu implementieren?

Allgemeines

- Die Aufgaben sind in Eclipse zu bearbeiten und beim Testat vorzuführen.
- Legen Sie für die Bearbeitung dieses Übungsblattes ein Paket namens `pr1.uebung10` an, in dem Sie Ihre Klassen anlegen.
- Erlaubt sind `MakeItSimple`-Funktionen (keine nicht besprochene Funktionalität aus der Java-Standard-Bibliothek) und das bisher erworbene Wissen aus den PR1-Vorlesungen. Sie müssen alle(!) von Ihnen verwendeten Konstrukte der Sprache sowie alle verwendeten Methoden, die nicht aus der Hilfsbibliothek `MakeItSimple` stammen, gut erklären und nötigenfalls im Testat selbst programmieren können!
- Sie geben ab, indem Sie vor Ende der Abgabefrist Ihr Projekt mit den lauffähigen Programmen des Übungsblattes in das Repository auf *ovid* pushen. Achten Sie darauf, ob Sie die Kontroll-E-Mail bekommen! Die letzte hochgeladene Version Ihres Projekts wird gewertet. Andere Abgaben, ob elektronisch oder auf Papier, zählen als **nicht abgegeben**!