Technische Hochschule Ingolstadt

Vorführaufgabe 8: die Abspielliste

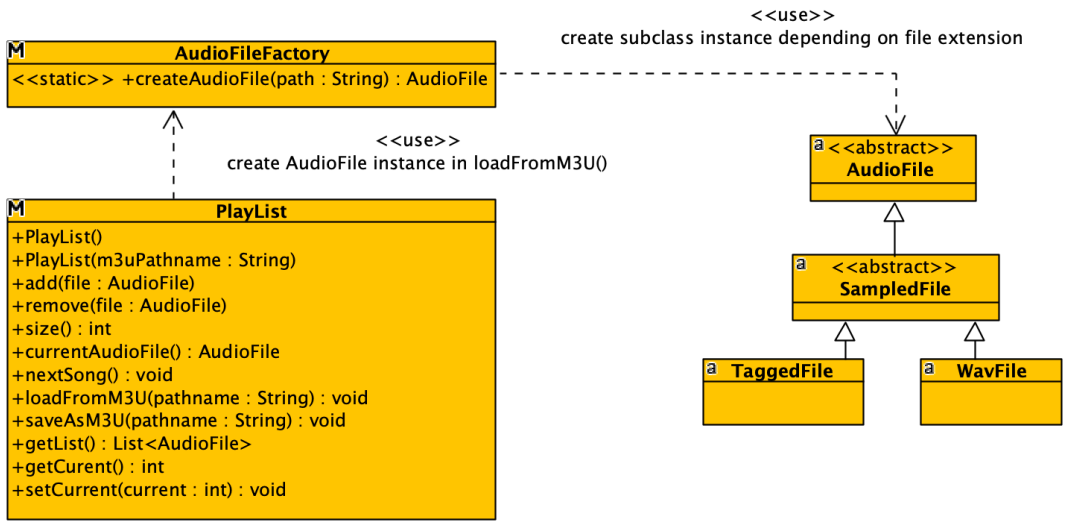
Ziel der Vorführaufgabe 8 ist die Implementierung einer Abspielliste (Play-Liste) zur Verwaltung der abzuspielenden Audiodateien. Hierfür soll die **Klasse PlayList** implementiert werden.

Eine `PlayList`-Instanz soll folgende Funktionalität bieten

- Speichern und Löschen von `AudioFile`-Instanzen in bzw. aus der Liste
- Aktuelle Abspielposition in der Liste:
  - Setzen und Lesen der aktuellen `AudioFile`-Instanz
  - Weiterschalten auf das nächste `AudioFile`
- Sequenzieller Abspielmodus
- Laden der Liste aus einer Textdatei im M3U-Format
- Speichern der Liste im M3U-Format

Eine M3U-Datei enthält Dateipfade von Audiodateien, einen pro Zeile. Beim Einlesen einer M3U-Datei muss entsprechend der Dateiextension der Audiodatei entweder ein `WavFile`- oder ein `TaggedFile`-Objekt erzeugt werden. Diese Aufgabe soll in eine einfache Implementierung einer sogenannten „Fabrikmethode“[1] übernehmen. Der Vorteil der Fabrikmethode liegt in der Zentralisierung der Objekterzeugung, wodurch Fehlermöglichkeiten reduziert werden: Kommen weitere `AudioFile`-Arten hinzu, so muss lediglich die `createAudioFile()`-Methode angepasst werden, was den Anpassungsaufwand minimiert.

Die nachfolgende Abbildung zeigt das finale Klassendiagramm.



[1] engl. factory method; siehe z.B. hier: https://de.wikipedia.org/wiki/Fabrikmethode

Vorbereitungen

Kopieren Sie ihr Projekt zur vorangegangenen Vorführaufgabe, entfernen Sie die Tests im cert-Ordner und ersetzen Sie die Tests im „tests“-Ordner durch die der aktuellen Vorführaufgabe (Archiv tests.zip). Erzeugen Sie weiterhin einen „Source Folder“ namens „examples“ und entpacken sie „examples.zip“ dorthin.

Teilaufgabe a) Speichern der AudioFiles in einer Liste

Bevor die Implementierung der Methoden der `PlayList`-Klasse beginnen können, ist die Frage zu klären, wo die AudioFile-Referenzen der `PlayList` gespeichert werden sollen. Grundsätzlich gibt es zwei Möglichkeiten: Vererbung – etwa durch Ableitung von `LinkedList` – oder Aggregation, d.h. Einführung eines Attributes vom Typ `LinkedList` oder einer anderen `List`-Implementierung.

Aufgrund der größeren Flexibilität der Aggrationsvariante sollen die `AudioFile`-Instanzen der Abspielliste in einer Collection vom Typ `LinkedList` gespeichert werden. Definieren Sie ein entsprechendes Attribut in der Klasse `PlayList` und implementieren Sie den entsprechenden Getter `getList()`.

Hinweise:

- Der Getter `getList()` dient hier in erster Linie zur Vereinfachung der JUNIT-Tests. Im nächsten Aufgabenblatt werden Sie einen Iterator entwickeln, der die Iteration über die enthaltenen AudioFiles einer `PlayList` ermöglicht. Der Getter kann dann entfallen.
- Der Ergebnistyp von `getList()` ist `List<AudioFile>`. Dies bedeutet, dass jede Objektreferenz zu einem Typ zurückgegeben werden kann, der das List-Interface mit Basistyp `AudioFile` implementiert. Wenn Sie z.B. eine `LinkedList<AudioFile>`-Instanz nutzen, ist diese Bedingung erfüllt, genauso wie bei einer `ArrayList<AudioFile>`-Instanz. Diese Spezifikation des Ergebnistyps der Methode lässt also mehr Flexibilität bzgl. der Implementierung zu.

Ergänzen Sie anschließend Methoden für die Verwaltung des Listeninhalts:

- `add(AudioFile file)`: Fügen Sie die übergebene Referenz in die Liste ein
- `remove(AudioFile file)`: Löscht das übergebene AudioFile aus der Liste
- `int size()`: Geben Sie die aktuelle Anzahl an AudioFiles, die in der PlayList verwaltet werden, zurück

Zum Testen Ihrer Implementierung können Sie die bereitgestellten Tests in der **Datei TestSubtaskA.java** nutzen.

Teilaufgabe b) Aktuelle Abspielposition und Weiterschalten zum nächsten Song

Die Klasse `PlayList` soll die aktuelle Position in der Abspielliste verwalten. Realisieren Sie die entsprechende Getter- und Setter-Methode zum Lesen bzw. Setzen der Position. Der parameterlose Konstruktor soll die Position geeignet initialisieren.

Realisieren Sie zudem die Methode `currentAudioFile()`, welche das `AudioFile`-Objekt an der aktuellen Position zurückliefert. Falls es keine gültige aktuelle Position gibt, d.h. der intern

verwaltete Index ungültig ist – soll die Methode `null` zurückgeben. Ein ungültiger Index ergibt sich zum Beispiel, wenn die Liste leer ist oder der Index auf einer Position steht, die nach dem Löschen von `PlayList`-Einträgen nicht mehr gültig ist.

Die Methode `nextSong()` soll die Abspielposition zum nächsten Lied weiterschalten. Falls das Listenende erreicht ist, soll die Abspielposition auf das erste Lied in der Liste (Position 0) gesetzt werden. Sollte der interne Index beim Aufruf der Methode auf einer ungültigen Position stehen, so soll der Index ebenfalls auf Position 0 gesetzt werden.

Zum Testen Ihrer Implementierung können Sie die bereitgestellten Tests in der **Datei TestSubtaskB.java** nutzen.

Mustercode zum Schreiben und Lesen von Textdateien

Die verbleibenden zwei Methoden `saveAsM3U()` und `loadFromM3U()` schreiben bzw. lesen Textdateien im M3U-Format. Vorbereitend hierfür finden Sie in der bereitgestellten Klasse `TextFileIO` (siehe examples.zip) ein Beispielprogramm, welches ein Array von Strings in eine Text-Datei schreibt und die erstellte Textdatei anschließend wieder einliest. Beim Einlesen wird jeder Zeile eine Zeilennummer vorangestellt. Das Hauptprogramm gibt die eingelesenen Strings abschließend auf die Konsole aus.

Erarbeiten Sie sich anhand der Kommentare in der Klasse `TextFileIO` und durch Ausführen bzw. Debuggen des Codes das nötige Grundverständnis für die Implementierung der beiden verbleibenden Methoden.

Teilaufgabe c) Speichern der PlayList im M3U-Format

Die Methode `saveAsM3U()` soll die `AudioFile`-Liste einer `PlayList` in einer Textdatei im M3U-Format speichern.

Eine M3U-Datei[2] besitzt die Extension „m3u“ und kann Kommentarzeilen, leere Zeilen oder Zeilen mit Dateipfaden für Audiodateien enthalten. Eine Kommentarzeile beginnt mit einem „#“, gefolgt von beliebigem Text. Eine leere Zeile enthält keine Zeichen bzw. nur Whitespace-Zeichen.

Das M3U-Fomat wird anhand des nachfolgenden Beispiels veranschaulicht:

```
# My best songs
Alternative\Band - Song.mp3

Classical\Other Band - New Song.mp3
Stuff.mp3
D:\More Music\Foo.mp3
..\Other Music\Bar.mp3
```

Obige Datei enthält einen Kommentar, eine Leerzeile und 5 Dateipfade.

Implementieren Sie nun die Methode `saveAsM3U(String pathname)`, welche die Dateipfade der

[2] Siehe http://en.wikipedia.org/wiki/M3U

in der Liste enthaltenen AudioFile-Instanzen in einer Datei mit dem durch `pathname` gegebenen Namen abspeichert.

Zum Testen Ihrer Implementierung können Sie die bereitgestellten Tests in der **Datei TestSubtaskC.java** nutzen.

Teilaufgabe d) Factory Method zum Erzeugen von AudioFile-Instanzen

Bisher wurden `AudioFile`-Instanzen (`TaggedFiles`, `WavFiles`) jeweils dort erzeugt, wo sie benötigt wurden. So praktisch diese Vorgehensweise vermeintlich ist, so fehleranfällig ist sie auch! Die Art der `AudioFile`-Instanz, welches zu erzeugen ist, hängt von der Extension des Dateinamens ab. Für ein korrekt laufendes Programm muss also stets der korrekte Konstruktor abhängig von der Extension des Dateinamens aufgerufen werden.

Zur Vermeidung derartiger Fehlerquellen wird die Erzeugung von `AudioFile`-Instanzen in einer Fabrikmethode zentralisiert. Die statische Methode `createAudioFile(String path)` erzeugt abhängig von der in `path` enthaltenen Extension die entsprechende `AudioFile`-Instanz und gibt diese zurück:

- Bei der Extension „wav“ wird eine `WavFile`-Instanz erzeugt.
- Bei Vorliegen der Extensionen „ogg“ bzw. „mp3“ wird eine `TaggedFile`-Instanz erzeugt.
- Liegt keine der oben genannten Extensionen vor, so soll eine `RuntimeException` mit einer sprechenden Fehlermeldung geworfen werden.
  Beispiel: `AudioFileFactory.createAudioFile("media/beispiel.txt")` sollte die `RuntimeException` mit dem Fehlertext ‚Unknown suffix for AudioFile "media/beispiel.txt"‘ werfen.

Hinweis: Ignorieren Sie für die Vergleiche der Dateiextension Groß- und Kleinschreibung.

Zum Testen Ihrer Implementierung können Sie die bereitgestellten Tests in der **Datei TestSubtaskD.java** nutzen.

Teilaufgabe e) Initialisieren einer PlayList aus einer M3U-Datei

Implementieren Sie nun die Methode `loadFromM3U(String path)`, welche die M3U-Datei mit dem übergebenen Dateipfad `path` öffnet und zu jedem in der Datei enthaltenen Audiodateipfad mithilfe der Fabrikmethode eine Audiodatei erzeugt. Die erzeugten Audiodateien sollen in der `PlayList` gespeichert werden.

Hinweis: Initialisieren Sie die `AudioFile`-Liste vor jedem Einlesevorgang. Das umfasst das Löschen bestehenden Listeneinträge sowie das Zurücksetzen der aktuellen Abspielposition.

Ergänzen Sie zu guter Letzt noch die korrekte Implementierung des zweiten Konstruktors, welcher den übergebenen Pfadnamen nutzt, um das Laden der PlayList aus einer M3U-Datei anzustoßen.

Zum Testen Ihrer Implementierung können Sie die bereitgestellten Tests in der **Datei TestSubtaskE.java** nutzen.

Hinweise zur Abnahme Ihrer Implementierung der Vorführaufgabe 08

Laden Sie alle zum Aufgabenblatt gehörigen Abnahme-Tests herunter und speichern Sie diese im Source-Folder `cert` Ihres Projekts. Dann führen Sie die Abnahme-Tests in Eclipse aus.

Wenn alle Tests ohne Fehler durchlaufen, mailen Sie bitte folgende Java-Dateien als Dateianhang mit dem Betreff „VA08“ an den APA-Server:

- AudioFile.java,
- AudioFileFactory.java,
- PlayList.java,
- SampledFile.java,
- TaggedFile.java,
- WavFile.java