P1: Spotify Ranking

Abgabetermin: 27.3.2025

Punktzahl: 37 + 3 Punkte (Einhaltung der Coding-Richtlinien)

In diesem Projekt sollen Sie eine Sammlung von Song-Daten verarbeiten und auswerten. Hierzu müssen Sie die Daten zunächst aus einer CSV-Datei einlesen, dann analysieren und die Ergebnisse über eine grafische Benutzeroberfläche (GUI) anzeigen. Die Anwendung umfasst mehrere Teilaufgaben, in denen Sie die Daten filtern, sortieren und in einem XML-Format speichern sollen.

Sie finden in ILIAS die Archivdatei `p1_vorgaben.zip` vor, die Sie in *Eclipse* als Projekt importieren können. Ändern Sie den Namen des Projekts dann in *IhrNachname.IhrVorname.P1* (rechte Maustaste auf den Projektnamen im Project Explorer, dann Refactor + Rename auswählen).

Vorgaben

Die Vorgaben unter `p1.zip` definieren bereits eine Grundstruktur, die Sie für Ihre Implementierung verwenden sollen. Darin sind mehrere Packages und ein Verzeichnis mit verschiedenen Ressourcen vorgegeben, die nachfolgend kurz erläutert werden.

Package `model`

Dieses Package enthält die Klasse zur Modellierung der Daten:

- **Song**: Diese Klasse repräsentiert einen einzelnen Song mit Attributen für Titel, Künstler, Veröffentlichungsjahr und Anzahl der Streams. Alle Attribute sind über Getter- und Setter-Methoden zugänglich. Die Klasse verfügt über einen Konstruktor und die von der Klasse `Object` geerbte Methode `toString()` ist überschrieben und gibt eine textuelle Repräsentation eines Songs zurück.

Package `exception`

Dieses Package ist zu Beginn noch leer. Sie müssen hier später eine Exception-Klasse erstellen.

Package `gui`

Die grafische Benutzeroberfläche wird in diesem Package definiert:

- **GUI**: Diese Klasse stellt die von Ihnen auzurufende Hauptklasse dar. Sie stellt eine interaktive Benutzeroberfläche bereit, um verschiedene Operationen wie das Anzeigen der Ergebnisse der von Ihnen implementierten Teilaufgaben auszuführen.

Package `data`

Dieses Verzeichnis enthält die für das Projekt erforderlichen Ressourcendateien:

- **spotify_songs.csv**: Diese Datei beinhaltet 953 Zeilen mit Informationen zu jeweils einem Song, welche die auszuwertenden Daten repräsentieren. Die folgende Tabelle beschreibt den Inhalt der Spalten in jeder Zeile.

Nr.	Spaltenname	Spaltenbeschreibung
1	`track_name`	Name des Songs
2	`artist(s)_name`	Name des/der Künstler(s)
3	`artist_count`	Anzahl der Künstler (Band zählt als 1)
4	`released_year`	Veröffentlichungsjahr
5	`released_month`	Veröffentlichungsmonat
6	`released_day`	Veröffentlichungstag
7	`in_spotify_playlists`	Anzahl der Spotify-Playlists, in denen der Song ist
8	`in_spotify_charts`	Platzierung in Spotify-Charts
9	`streams`	Anzahl der Spotify Streams
10	`in_apple_playlists`	Anzahl der Apple-Playlists, in denen der Song ist
11	`in_apple_charts`	Platzierung in Apple-Charts
12	`in_deezer_playlists`	Anzahl der Deezer-Playlists
13	`in_deezer_charts`	Platzierung in Deezer-Charts, in denen der Song ist
14	`in_shazam_charts`	Platzierung in Shazam-Charts
15	`bpm`	Beats per Minute

Tabelle 1: Inhalt der Spalten der Spotify Songs CSV-Datei

- **playlist.dtd**: Definiert das XML-Format für Playlists. Die oberste Ebene eines XML-Dokuments, welches dieser DTD entspricht, besteht aus einem `playlist`-Element. Dieses Element besitzt das Attribut `name`, das den Namen der Playlist angibt. Innerhalb des `playlist`-Elements müssen ein oder mehrere `song`-Elemente vorkommen. Ein `song`-Element besteht aus genau vier untergeordneten Elementen, und zwar in folgender Reihenfolge:
  - `title`: Enthält den Titel des Songs.

- `artist`: Enthält den Namen des Künstlers.
- `year`: Gibt das Erscheinungsjahr des Songs an.
- `streams`: Beschreibt die Anzahl der Streams des Songs.

Alle diese Elemente enthalten ausschließlich Informationen, die in Form von Text gegeben sind (PCDATA).

- **test_playlist.xml**: Diese Datei enthält eine Beispiel-Playlist im XML-Format, mit der überprüft werden kann, ob Ihre Implementierung des XML-Readers richtig funktioniert.
- **my_own_playlist.xml**: Diese Datei enthält eine Playlist im XML-Format, welche erst beim Testen Ihrer Implementierung von Aufgabe 3b generiert wird.
- **all_songs.xml**: Diese Datei beinhaltet alle (korrekt beschriebenen) Songs aus der CSV Datei im XML-Format.

Package `utils`

In diesem Package befinden sich Klassen zur Verarbeitung der Daten:

- **CSVReader**: Mit Hilfe dieser Date soll später die CSV-Datei eingelesen und eine Liste von `Song`-Objekten erstellt werden. Dabei sollen fehlerhafte Zeilen übersprungen und Ausnahmen behandelt werden. Die Methoden hierzu sowie eine Exception-Klasse sind von Ihnen in Aufgabe 1 zu implementieren.
- **DataFilter**: Diese Klasse beinhaltet Methoden zur Filterung und Analyse der Song-Daten, welche von Ihnen in Aufgabe 2 implementiert werden müssen.

Package `xml`

Dieses Package enthält Klassen zum Lesen und Schreiben von Playlists in einem (vorgegebenen) XML-Format:

- **XMLReader**: Mit Hilfe dieser Klasse soll eine XML-Datei eingelesen und eine Liste von `Song`-Objekten erstellt werden. Die Methode `readPlaylistFromXML()` parst eine XML-Datei und erstellt ein XML-Dokument. In Aufgabe 3a müssen Sie daraus eine Songliste erstellen und zurückgeben.
- **XMLWriter**: Diese Klasse dient dazu eine Playlist in einer XML-Datei zu speichern. Die Methode `writePlaylistToXML()` generiert ein XML-Dokument (mit der Wurzel) und speichert dieses in einer Datei. In Aufgabe 3b müssen Sie die Daten aus einer Song-Sammlung in Form von passenden XML-Elementen erstellen und in das Dokument einfügen,

- **XML_PushParser** Diese Klasse generiert einen Sax Parser, mit dessen Hilfe eine XML-Datei eingelesen werden kann. Dazu wird ein Content Handler gesetzt, mit dessen Hilfe Informationen aus dem zugehörigen XML-Dokument gefiltert werden können.
- **PlayListContentHandler** Dieser Content Handler dient dazu, während des Parse-Prozesses Informationen aus dem XML-Dokument zu filtern.

Quelle

Der hier verwendete Datensatz beruht auf https://www.kaggle.com/datasets/abdulszz/spotify-most-streamed-songs (Stand: 07.09.2024).

Aufgabe 1: CSVReader (5 P)

Zuerst sollen Sie die Daten aus der CSV-Datei einlesen. Ergänzen Sie hierzu die Klasse `p1.utils.CSVReader` wie im Folgenden beschrieben:

(a) parseFile (2 P)

Implementieren Sie die Methode `parseFile`, die eine CSV-Datei mit dem als Parameter übergebenen Dateipfad zeilenweise einlesen soll. Dabei sollen folgende Schritte berücksichtigt werden:

- Erstellen Sie zu Beginn der Methode eine `ArrayList` für `Song`-Objekte.
- Öffnen Sie die CSV-Datei mit einem `try-with-resources`-Konstrukt und verwenden Sie einen `BufferedReader` und übergeben Sie diesem bei der Erstellung den `filepath`.
- Die erste Zeile der Datei enthält die Spaltenüberschriften und soll daher übersprungen werden.
- Lesen Sie jede weitere Zeile der Datei ein und übergeben Sie diese der Methode `parseLine` zur Verarbeitung. Fügen Sie jedes von `parseLine` zurückgegebene `Song`-Objekt in die zuvor erstellte `ArrayList<Song>` ein.
  Hinweis: Die Methode `parseLine` kann `null` zurückgeben. In diesem Fall soll das `Song`-Objekt übersprungen, d.h. nicht in die `ArrayList` eingefügt werden.
- Geben Sie die, von Ihnen erstellte, `ArrayList` am Ende der Methode zurück.

b) parseLine (3 P)

Implementieren Sie die Methode `parseLine`, die eine einzelne Zeile aus der CSV-Datei verarbeitet, in ein `Song`-Objekt umwandelt und dieses zurückgibt. Falls die Zeile ungültig ist, soll `null` zurückgegeben werden. Gehen Sie wie folgt vor:

- Zerteilen Sie die übergebene Zeile anhand des (Spalten-)Trennzeichens (`";"`) in ein Feld von Tokens.
- Überprüfen Sie, ob die übergebene Zeile eine passende Zahl von Tokens hat (d.h. ob die Anzahl der Tokens der Anzahl der definierten Spalten entspricht). Ist dies nicht der Fall, geben Sie `null` zurück.
- Erstellen Sie, falls möglich, ein neues `Song`-Objekt mit den relevanten Informationen aus den Tokens. Prüfen Sie dabei, ob die Tokens für `streams` und `released_year` korrekt geparst werden können. Sollte beim Parsen ein Fehler auftreten, fangen Sie die zugehörige `NumberFormatException` ab, geben Sie eine Fehlermeldung aus und geben Sie dann `null` zurück.
- Fall ein `Song`-Objekt korrekt erstellt werden konnte, geben Sie dieses am Ende zurück.

Nach erfolgreicher Umsetzung von Aufgabe 1 (`a` und `b`) können Sie die GUI starten, indem Sie die Klasse `p1.gui.GUI` ausführen. Beim Laden der Daten aus der CSV-Datei müssten in der Konsole dabei zwei NumberFormatExceptions angezeigt werden.

Aufgabe 2: Daten filtern (22 P)

Im Folgenden müssen Sie die Klasse `p1.utils.DataFilter` ergänzen. Eine Sammlung aller Song-Objekte, die Sie hierzu filtern müssen, ist als Attribut in der Klasse vorgegeben.

a) IllegalSongException (1 P)

Erstellen Sie in dem bisher leeren Package `p1.exception` eine `IllegalSongException`-Klasse, die von `Exception` erbt. Implementieren Sie einen Konstruktor, der die Fehlermeldung als Parameter entgegennimmt und diese an die Oberklasse weiterleitet. Nach Beendigung dieser Teilaufgabe sollen Sie in der `DataFilter`-Klasse den auskommentierten Import der `IllegalSongException`-Klasse einkommentieren (Zeile 10).

b) SongComparator (4 P)

Erstellen Sie nun in der `DataFilter`-Klasse eine private innere `SongComparator` Klasse, die das Interface `Comparator` implementiert und zwei `Songs` miteinander vergleicht. Diese Klasse soll im Konstruktor einen Parameter vom Typ `String` entgegennehmen,

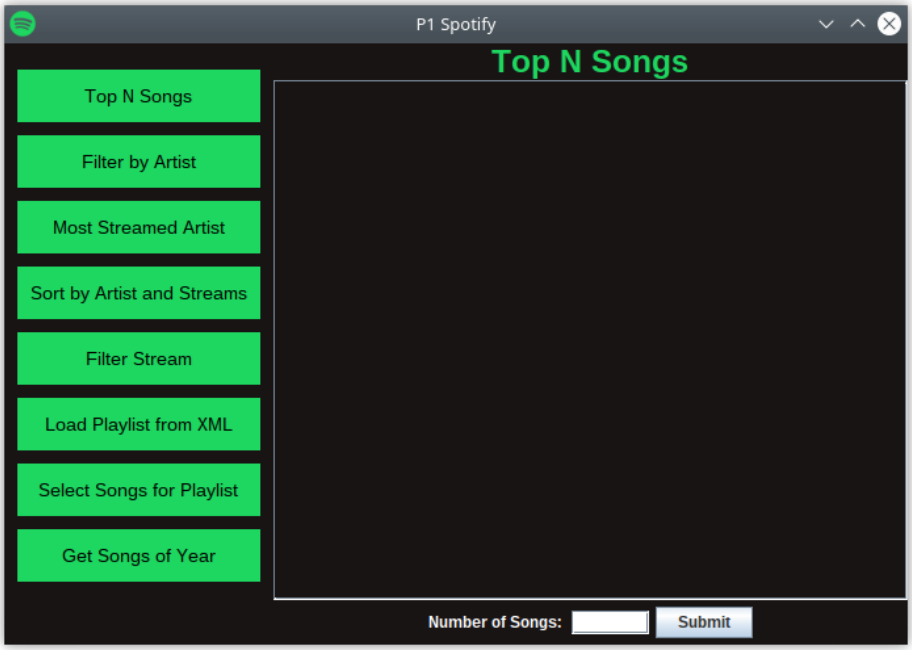

Abbildung 1: (Aufgabe 1): Start GUI

der angibt, nach welchem Attribut die Songs verglichen werden sollen (*title*, *artist*, *year* oder *streams*). Der Parameterwert soll in einer Instanzvariablen gespeichert werden, um in der `compare`-Methode verwendet zu werden. Wandeln Sie den Parameter-String dabei in Kleinbuchstaben um, so dass es egal ist, ob dem Konstruktor der Wert in Groß- oder Kleinschreibung übergeben wird. Überschreiben Sie die `compare`-Methode so, dass sie mit Hilfe eines `switch`-Statements die Songs basierend auf dem übergebenen Parameter vergleicht.

- title und artist: Hier soll nach der alphabetischen Reihenfolge verglichen werden (Groß-und Kleinschreibung soll ignoriert werden).
- year und streams: Hier soll nach der numerischen Größe verglichen werden, wobei bei `year` ältere Jahre vor neueren Jahren und bei `streams` größere Werte vor kleineren Werten stehen sollen.

Sollte ein ungültiger Parameter übergeben werden, werfen Sie eine `IllegalArgumentException` mit einer passenden Fehlermeldung.

c) Top N Songs filtern (2 P)

Implementieren Sie die Methode `getTopNSongs`, die eine Liste der `n` besten Songs zurückgibt, basierend auf der Anzahl an Streams. Gehen Sie dabei wie folgt vor:

- Erstellen Sie ein Comparator-Objekt der zuvor in Teilaufgabe `2b` erstellten Klasse, wobei als Sortierkriterium *"streams"* verwendet wird.
- Sortieren Sie den Inhalt der Liste `songs` mithilfe einer passenden Methode und dem zuvor erstellen `SongComparator`.
- Erstellen Sie nun eine neue Liste mit den ersten `n` Elementen aus der Liste `songs` (sollte `n` größer als die Anzahl der vorhandenen Songs in der Instanzvariable sein, nehmen Sie die Größe der Liste als Wert für `n`). Hinweis: Sie können eine Methode aus dem Interface `List` verwenden, um eine passende Teilliste zu generieren.
- Geben Sie die gefüllte Liste am Ende als Ergebnis zurück.

d) Songs eines Künstlers filtern (2 P)

Nun sollen Sie die Methode `filterSongsByArtist` implementieren, welche eine Liste von allen Song-Objekten eines Künstlers zurückgibt, welcher durch einen Teilstring des Künstlernamens identifiziert wird, welcher der Methode als Parameter übergeben wird. Implementieren Sie die Methode wie folgt:

- Legen Sie zunächst eine Liste an, welche die Songs des Künstlers enthalten soll.
- Wandeln Sie den als Parameter übergebenen String in einen String mit ausschließlich Kleinbuchstaben um.
- Durchlaufen Sie die Liste `songs` mit einer foreach-Schleife und überprüfen Sie für jeden Song, ob der als Parameter übergebene und in Kleinbuchstaben umgewandelte String als Teilstring in dem `artist` Attribut des Songs vorkommt. Wandeln Sie den Wert des `artist` Attributs dabei immer vor der Überprüfung in Kleinbuchstaben um, damit der Vergleich unabhängig von Groß- und Kleinschreibung ist.
- Ist der Wert des Parameters in dem `artist` Attribut des Song-Objekts enthalten, soll das `Song`-Objekt der von Ihnen erstellten Liste hinzugefügt werden.
- Am Ende soll die Liste zurückgegeben werden.

e) Künstler mit den meisten Streams (4 P)

Implementieren Sie eine Methode, die den Künstler mit der höchsten Gesamtzahl an Streams aus der Instanzvariable ermittelt und eine entsprechende Beschreibung als String zurückgibt. Gehen Sie dabei wie folgt vor:

- Erstellen Sie eine `Map`, bei der der Künstlername als Schlüssel und die gesamte Anzahl der Streams als Wert gespeichert wird.

- Durchlaufen Sie die Liste der Songs und addieren Sie die Streams jedes Songs zum entsprechenden Künstler in der Map. Falls der Künstler noch nicht in der Map vorhanden ist, fügen Sie ihn mit der Anzahl der Streams des aktuellen Songs hinzu.
- Durchlaufen Sie die Einträge in der Map, um den Künstler mit der höchsten Gesamtanzahl an Streams zu finden.
- Geben Sie eine Beschreibung des meistgestreamten Künstlers zurück, im Format: `[Künstlername] with [Anzahl der Streams] streams`. Die Anzahl der Streams soll dabei mir der Konstanten `GUI.dfStreams` formatiert werden.

f) Liste nach Künstler und Streams filtern (3 P)

Implementieren Sie die Methode `sortSongsByArtistAndStreams`, die eine Liste von Songs zuerst nach Künstlern (alphabetisch) und dann bei gleichen Künstlern nach Streams (absteigend) sortiert und zurückgibt. Gehen Sie dabei wie folgt vor:

- Erstellen Sie je eine `SongComparator`-Instanz, welche die Songs alphabetisch nach dem Künstler bzw. nach der Anzahl der Streams sortiert. In letzterem Fall sollen Songs mit mehr Streams vor Songs mit weniger Streams stehen (numerisch absteigende Sortierung)
- Erstellen Sie eine Liste, welche Sie mit allen `Song`-Objekten aus der Liste `songs` füllen.
- Sortieren Sie diese Liste mit Hilfe der Methode `sort`. Verwenden Sie als Parameter einen Lambda-Ausdruck, der zwei `Song`-Objekte wie folgt vergleicht:
  - Vergleichen Sie die Künstler der beiden `Song`-Objekte mit Hilfe des ersten `Comparator`-Objekts, welches Sie im ersten Schritt erstellt haben.
  - Sollten zwei Songs denselben Künstler haben, vergleichen Sie die beiden Song-Objekte ggf. mit dem zweiten `Comparator`-Objekt, welches Sie im ersten Schritt erstellt haben.
  - Der Ergebniswert sollte kleiner (größer) 0 sein, wenn der erste Song in der Sortierung vor (nach) dem zweiten Song kommt.
- Geben Sie die sortierte Liste am Ende zurück.

g) Weitere Informationen (6 P)

Hier sollen Sie nun noch drei Methoden implementieren bzw. vervollständigen, mit deren Hilfe weitere Informationen aus der Liste der Songs gefiltert und in einem Ergebnistext gespeichertwerden:

- die Anzahl der Songs in der Liste der meistgestreamten Songs, die jeweils aus einer Dekade stammen (1991 - 2000, 2001 - 2010, 2011 - 2020, ab 2021)
- eine Liste der meistgestreamten Künstler

Hierbei sollen Sie jeweils Stream-Verarbeitung und Lambda-Expressions verwenden.

Gehen Sie wie folgt vor:

- Die Methode `numSongsInYears` soll die Anzahl der Songs in der Liste `songs` bestimmen, die aus einem Jahr stammen, welches im Intervall $[a, b-1]$ liegt. Die Werte von a und b werden der Methode dabei als Parameter übergeben. Erzeugen Sie hierzu aus der Liste `songs` einen Stream, filtern Sie dann in die Songs aus, die aus einem Jahr stammen, welches in dem genannten Intervall liegt und wenden Sie dann eine Abschlussoperation auf den Stream an, in der Sie die Anzahl der Objekte in dem gefilterten Stream bestimmen. Geben Sie das Ergebnis in Form eines Strings zurück, welcher das Intervall und die Anzahl der Sóngs beinhaltet und wie in folgendem Beispiel aussieht:

  ```
  1991 -> 2000: 14 songs
  ```

- Die Methode `topNSongArtists` soll einen Text mit einer Liste von Strings zurückgeben, die die n meist gestreamten Songs beschreiben (durch die jeweiligen Künstler und den Titel des Songs). Erzeugen Sie hierzu aus der Liste `songs` einen Strom und sortieren Sie diesen absteigend nach der Anzahl der Streams. Verwenden Sie hierbei einen Lambda-Ausdruck und die Methode `Long.compare` Begrenzen Sie den Strom dann auf die ersten `n` Elemente und ersetzen Sie jedes `Song`-Objekt im Strom durch einen String, der aus dem Wert des Attributs `artist` besteht, gefolgt von dem Wert des Attributs `title` in runden Klammern. Als Abschlussoperation hängen Sie jeden String in dem Strom, zusammen mit einem Zeilenumbruchszeichen an den Inhalt der vorgegebenen `StringBuffer` Instanz an.
- In der Methode `filterStreamForInfos` werden die Ergebnisse aus den beiden Methoden `numSongsInYears` und `topNSongArtists` in einer `StringBuffer` Instanz zusammengetragen, dessen Inhalt am Ende als Ergebnis zurückgegeben wird. Ergänzen Sie die Methode `filterStreamForInfos` so, dass die Methode `numSongsInYears` in einer Schleife jeweils für die Werte $(a = 1991, b = 2001)$, $(a = 2001, b = 2011)$, $(a = 2011, b = 2021)$ und $(a = 2021, b = 2031)$ aufgerufen und die Ergebnisse an den Inhalt der vorgegebenen `StringBuffer` Instanz angehängt werden.

  Rufen Sie dann noch die Methode `topNSongArtists` für den vorgegebenen Wert von n auf und hängen Sie das Ergebnis ebenfalls an den Inhalt der vorgegebenen `StringBuffer` Instanz an.

Sie können nach den Teilaufgaben 2 (`c-g`) mit Hilfe der GUI überprüfen, ob Ihre Implementierung korrekt ist (siehe Screenshots auf den Seiten 12-14).

Aufgabe 3: XML (10 P)

In dieser Aufgabe sollen Sie die Methoden `readPlaylistFromXML` und `writePlaylistToXML` in den Klassen `XMLReader` und `XMLWriter` vervollständigen.

a) XMLReader (3 P)

Ergänzen Sie die Methode `readPlaylistFromXML` in der Klasse `XMLReader`. Die Implementierung soll Informationen zu einer Playlist aus einer XML-Datei auslesen und in eine Liste von `Song`-Objekten umwandeln. Gehen Sie wie folgt vor:

- Holen Sie sich vom bereits vorgegebenen `Document` das Wurzelelement.
- Erstellen Sie eine neue `ArrayList` für `Song`-Objekte.
- Durchlaufen Sie mithilfe einer Schleife alle Kindelemente des Wurzelelements vom Element-Typ `song`.
- Für jedes Kindelement vom Typ `song` extrahieren Sie die Werte der Unter-Elemente (`title`, `artist`, `year` und `streams`).
- Verwenden Sie diese Werte, um ein neues `Song`-Objekt zu erstellen und fügen Sie dieses der von Ihnen erstellten Liste hinzu.
- Geben Sie am Ende die gefüllte Liste zurück.

Um Ihre Implementierung zu überprüfen, klicken Sie zunächst auf den Button *"Load Playlist from XML"* und anschließend auf den Button *"Test XMLReader with Example File"*. Daraufhin erscheint ein `"Success"`-Pop-up-Fenster und der Inhalt der Datei wird in einer Tabelle angezeigt (siehe Abbildung 7, Seite 15).

b) XMLWriter (3 P)

Ergänzen Sie die Methode `writePlaylistToXML` in der Klasse `XMLWriter`. Die Methode soll die Informationen aus einer als Parameter übergebenen `Playlist` in ein XML-Format umwandeln und speichern. Das Wurzelelement `rootElement`, welches die Playlist repräsentiert, ist bereits vorgegeben. Gehen Sie wie folgt vor:

- Durchlaufen Sie die übergebene Liste `playlist`-Instanz und erstellen Sie für jeden darin enthaltenen Song ein neues `Song`-Element.
- Für jedes Attribut des aktuellen Songs (`title`, `artist`, `year` und `streams`) erstellen Sie jeweils ein eigenes XML-Element.
- Entnehmen Sie die benötigten Informationen aus dem `Song`-Objekt und fügen Sie die Werte den entsprechenden XML-Elementen hinzu.

- Hängen Sie alle erstellten Attribut-Elemente an das `Song`-Element an.
- Fügen Sie abschließend das `Song`-Element dem Wurzelelement `rootElement` hinzu.

Überprüfen Sie Ihre Implementierung, indem Sie im Bereich *"Select Songs for Playlist"* mehrere Lieder auswählen und anschließend auf den *"Save Playlist (my_ own_ playlist.xml)"* Button klicken. Die Playlist wird dann in die Datei `my_own_playlist.xml` geschrieben. Kontrollieren Sie danach den Inhalt dieser Datei (die im `data`-Verzeichnis liegt), um sicherzustellen, dass die von Ihnen ausgewählten Songs korrekt gespeichert wurden.

c) PlaylistContentHandler (4 P)

Die Klasse `PlaylistContentHandler` implementiert das `ContentHandler` Interface. Die Methoden des Interface werden vom SAX Parser aufgerufen, der in der `XML_PushParser` Klasse generiert wird und der dazu verwendet wird, eine XML-Datei mit einer Playlist zu parsen. Sie sollen die Klasse `PlaylistContentHandler` nun so erweitern, dass alle Songs aus einem vorgegebenen Jahr aus derPlaylist gefiltert und in einer Liste gespeichert werden, welche dann durch die Methode `getResult` zurückgegeben wird. Hierzu ist in der Klasse Folgendes vorgegeben:

- Die Instanzvariable `selYear` wird durch den vorgegebenen Konstruktor mit dem Jahr initialisiert, aus dem die Songs stammen, die herausgefiltert werden sollen.
- Die Instanzvariable `songs` wird dazu benötigt, die Songs aus dem vorgegebenen Jahr zu speichern.
- Der Wert der Instanzvariablen `charSequence` wird durch die vorgegebene Methode `characters` während des Parse-Vorgangs wiederholt auf Strings gesetzt, die aus zusammenhängenden Textstücken in dem geparsten XML-Dokument bestehen. Hierzu gehört in diesem Fall explizit der Text, der als Inhalt der XML-Elemente *title*, *artist*, *year* und *streams* in dem Dokument zu finden ist.
- Die Methode `getResult` gibt die Liste mit den gesammelten Songs zurück.
- Neben der Methode `characters` sind alle weiteren Methoden des `ContentHandler` Interface leer implementiert vorgegeben.

Ergänzen Sie die Klasse `PlaylistContentHandler` nun wie folgt:

- Legen Sie vier (private) Instanzvariablen an, in denen Sie später für jeden Song die Werte der XML-Elemente *title*, *artist*, *year* und *streams* speichern.
- Ergänzen Sie die Methoden `startElement` bzw. `endElement`, so, dass beim Parsen der XML-Datei
    - die Text-Inhalte der Elemente *title*, *artist*, *year* und *streams* in den zuvor angelegten Instanzvariablen gespeichert werden.

- Wenn ein einzelner Song vollständig gepased wurde, prüfen Sie, ob das Jahr, aus dem der Song stammt, dem Wert der Instanzvariablen `selYear` entspricht. Ist dies der Fall, erzeugen Sie eine Instanz der Klasse Song und speichern Sie diese in der Liste `songs`.

Hinweis: In dieser Aufgabe benötigen Sie nur eine der beiden Methoden. Die andere kann leer bleiben.

Abbildung 2: (Teilaufgabe 2c): Top N Songs mit n = 16

Abbildung 3: (Teilaufgabe 2d): Filter By Artist mit Beispiel: Drake

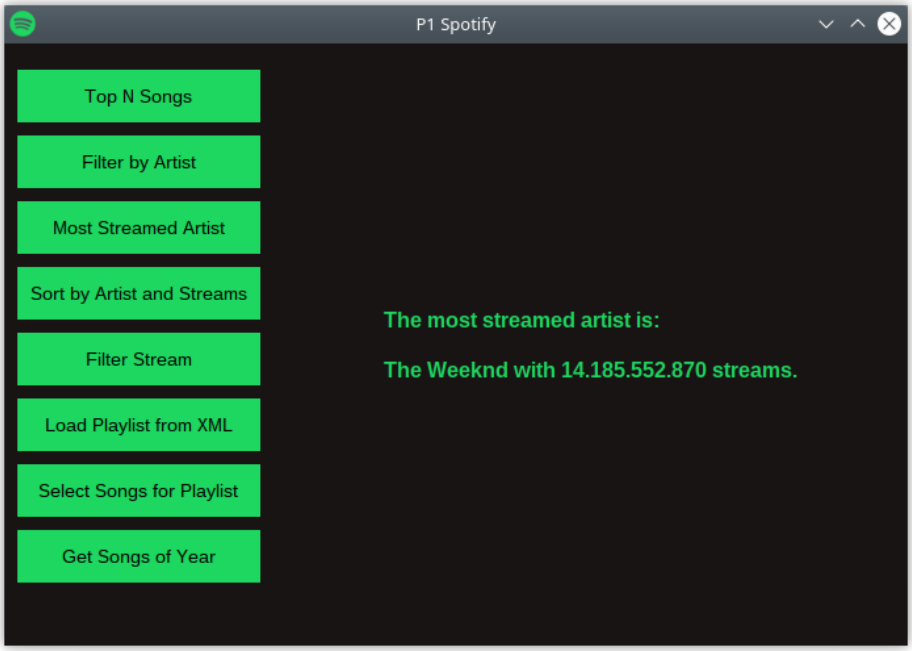

Abbildung 4: (Teilaufgabe 2e): Der meistgestreamte Künstler

Abbildung 5: (Teilaufgabe 2f): Beginn der sortierten Liste

Abbildung 6: (Teilaufgabe 2g): Anzahl Songs nach Dekaden und Künstler der meistgestreamten Songs

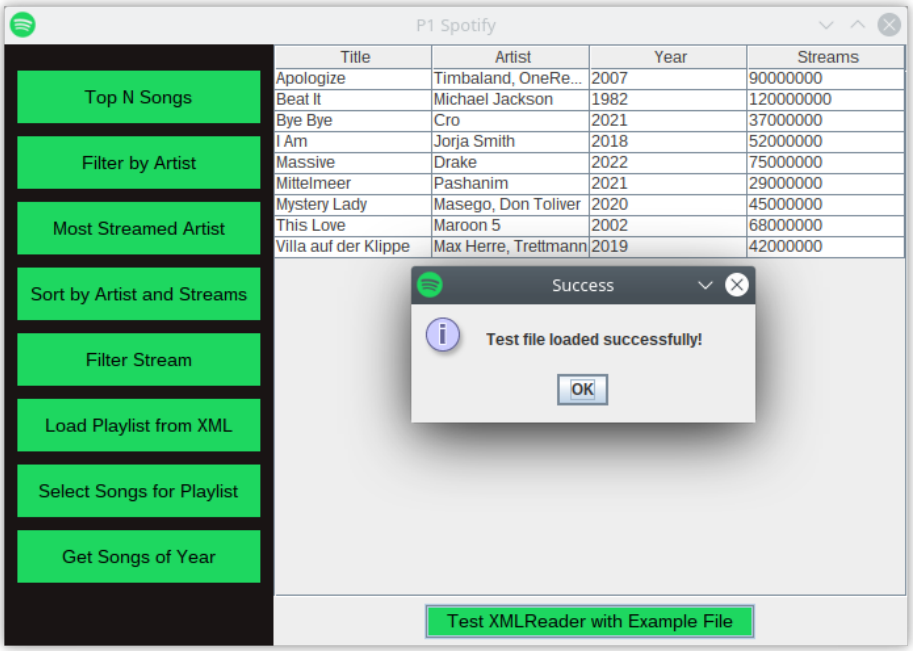

Abbildung 7: (Teilaufgabe 3a): Korrekte Darstellung der `test_playlist.xml`

.

Abbildung 8: (Teilaufgabe 3c): Songs aus dem Jahr 2000 (aus dem XML File gefiltert)

.