Praktikum 2 zu KMPS

Bei diesem Praktikumsversuch werden Sie mit ihrem Java-Programm aus dem letzten Versuch weiterarbeiten, und es in ein Scala-Programm überführen. Dabei werden Sie weitere funktionale Konzepte anwenden. Sollten Sie im letzten Praktikumsversuch keine Untermethoden bei Aufgabe 3 verwendet haben, werden Sie ihr Programm umstrukturieren müssen.

Aufgabe 1: (Scala Entwicklungsumgebung)

a. Wählen Sie eine für Sie möglichst geeignete (Entwicklungs-)Umgebung für Scala aus. Im Folgenden wird eine Möglichkeit vorgestellt (unter Verwendung der IDE *IntelliJ IDEA Community):*

   Laden Sie sich die aktuelle Version von IntelliJ IDEA Community auf der IntelliJ Download-Seite herunter und installieren Sie es.
   Bei dem ersten Start von IntelliJ können Sie, nach dem Festlegen eines UI-Themes und Download von Default-Plugins, im Abschnitt *Featured Plugins* das Plugin für Scala herunterladen.
   Erstellen Sie nun ein neues Projekt. Wählen Sie „Scala" und dann „IDEA" aus und klicken Sie auf „Next". Im Folgenden Dialog müssen Sie ein JDK und Scala SDK angeben. Falls Sie bereits ein JDK installiert haben, können Sie es mit einem Klick auf „New…" auswählen. Um das Scala SDK zu installieren, klicken Sie auf „Create…" dann „Download…" und bestätigen Sie die Version.
   Legen Sie in ihrem Scala Projekt links in dem Projekt-Fenster im „src" Ordner eine neue Datei vom Typ Scala Worksheet an. In einem Scala Worksheet werden alle Scala-Befehle von oben nach unten ausgeführt und ihr Wert zurückgegeben. Sie können dort auch Funktionen deklarieren.

b. Legen Sie zwei case-Klassen Track und Album wie folgt an:

```
case class Track(title: String, length: String, rating:
Int, features: List[String], writers: List[String])
case class Album(title: String, date: String, artist:
String, tracks: List[Track])
```

Einschränkungen bei der Programmierung:
Alle Funktionen sollen, wie auch beim ersten Praktikumsversuch, als *reine Funktionen* implementiert werden und dürfen keine Iterationen/Schleifen verwenden. Ausnahmen werden in den Aufgabenstellungen angegeben. Außerdem ist in den Praktikumsaufgaben *1 - 3* als Kontrollfluss-Konstrukt nur *Patternmatching* erlaubt (d.h. kein if/else o.ä.). Alle Funktionen müssen auf oberster Ebene aus einem Patternmatching-Ausdruck bestehen, d.h. nach der Angabe des Funktionstyps muss sofort nach dem Gleichheitszeichen ein Patternmatching-Ausdruck beginnen.

Beispiel:

```
def isOne(zahl: Int): Boolean = zahl match{
  case 1 => true
  case _ => false // „Unterstrich" matcht für alle Werte
}
```

Praktikum 2 zu KMPS

Sie dürfen in ihrem Code nur Konstanten (`val`) verwenden (und somit keine Variablen (`var`)).
Zum Bearbeiten von Listen dürfen Sie sowohl Appending (z.B. *mylist = mylist:+10*) als auch Prepending verwenden (z.B. *mylist = elem :: mylist*). Hierzu finden Sie auch weitere Erklärungen und Beispiele u.a. in der offiziellen Scala-Dokumentation. Zum Umwandeln von Strings zu Integer dürfen Sie die Funktion toInt verwenden.

Ansonsten ist das Benutzen von (weiteren) Funktionen der Scala-/Java-Standardbibliothek und Methoden von Objekten nicht erlaubt, es sei denn die Aufgabenstellung macht explizit eine Ausnahme.

Aufgabe 2: (Token-Liste erstellen)
Implementieren Sie zunächst die Funktionalität Ihrer Java-Main-Methode und ihrer Methode createTokenList in Scala in folgender Weise:

a. Lesen Sie die XML Datei mit `Source.fromFile("alben.xml").toList` in eine *Liste von Charactern* ein.
b. Implementieren Sie die Funktion `createTokenList`. Sie soll die gleiche Funktionalität wie Ihr Java-Pendant bieten.

Aufgabe 3 (Parsing der Token)
Schreiben Sie eine Methode `parseFile`. Sie soll die gleiche Funktionalität wie Ihr Java-Pendant bieten.

a. Schreiben Sie hierfür zwei Unterfunktionen `parseAlbum` und `parseTrack`, die das Parsen eines Albums bzw. Tracks übernehmen sollen, sobald aus dem aktuellen Token ersichtlich wird, dass ein Album bzw. Track folgt.
b. Rufen Sie die Funktion `parseFile` mit der Token-Liste auf und geben Sie den Rückgabewert (Liste von Alben) mit `println` aus.

Hinweise**:**

1. Um eine Kopie eines Objekts einer Case-Class mit einem geänderten Wert zu erhalten, dürfen Sie die Methode `copy` benutzen. Beispiel:
```
val track = Track("alter titel", "", 0, List(), List())
val trackMitNeuemTitel = track.copy(title = "neuer titel")
```
2. Ggf. brauchen Sie weitere „case clauses" in einem Patternmatching-Ausdruck, die z.B. den Vergleich auf (Daten-)Typen erlauben oder zusätzlich *ODER-Verknüpfungen* für mehrere Möglichkeiten (innerhalb einer „case clause") zulassen.