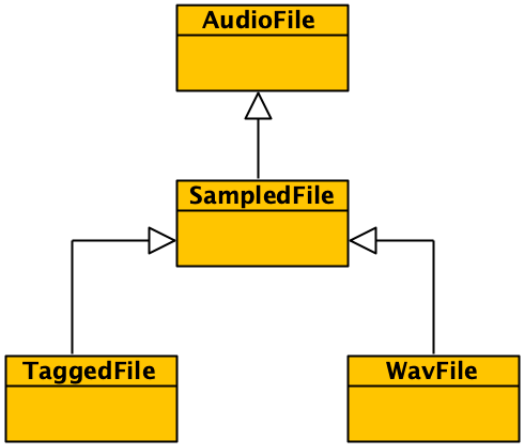
Technische Hochschule Ingolstadt

Vorführaufgabe 7: Vererbungshierarchie

Ziel dieser Vorführaufgabe ist es, die rechts abgebildete Klassenhierarchie und die Basisfunktionalitäten des Audioplayers zu implementieren.

Die Basisklasse `AudioFile` wurde bereits in Aufgabe 6 erstellt. Von ihr wird eine Klasse `SampledFile` abgeleitet, die gesampelte Audiodateien - etwa MP3, OggVorbis oder RIFF WAVE- repräsentiert. Die Klasse greift zum Dekodieren der Audiodateien auf in externen Java-Archiven (JAR-Dateien) enthaltene Klassen zurück.



In unserem Projekt werden von `AudioFile` außer `SampledFile` keine weiteren Klassen abgeleitet. Streng genommen könnte man also die beiden Klassen zu einer zusammenfassen. Es gibt aber Audiodateien wie z.B. MIDI, die nicht gesampelt sind und daher sauberer direkt von `AudioFile` abgeleitet werden würden. Aus diesem Grund nehmen wir diese Trennung vor.

Die von `SampledFile` abgeleitete Klasse `TaggedFile` repräsentiert Audiodateien mit Meta-Daten wie etwa ID3-Tags bei MP3 oder Vorbis Comments bei OggVorbis. Die andere von `SampledFile` abgeleitete Klasse repräsentiert RIFF WAVE-Dateien, die keine Metainformationen in Form von Tags bereitstellen. Solche Dateien erhält man beispielsweise, wenn man CDs mit einem entsprechenden Tool "rippt".

Das obige Klassendiagramm stellt die finale Version der Klassenhierarchie dar, in der die Funktionalität zur Nutzung von Audio-Dateien über die einzelnen Klassen folgendermaßen verteilt ist (siehe auch Klassendiagramm auf der nächsten Seite):

Klasse AudioFile
- Bietet Methoden zum Parsen des Dateinamens der Audiodatei.
- Gibt abstrakte Methoden zur Wiedergabe der Audiodatei, zum Pausieren bzw. Anhalten der Wiedergabe der Audiodatei sowie zur Formatierung der Abspieldauer und Abspielposition vor.

Klasse SampledFile

- Implementiert die in AudioFile vorgegebenen abstrakten Methoden.

Klasse TaggedFile

- Liest Dateieigenschaften aus den Tags der Audiodatei.

Klasse WavFile

- Liest Dateieigenschaften aus der Metainformation der Audiodatei.

Vorbereitungen

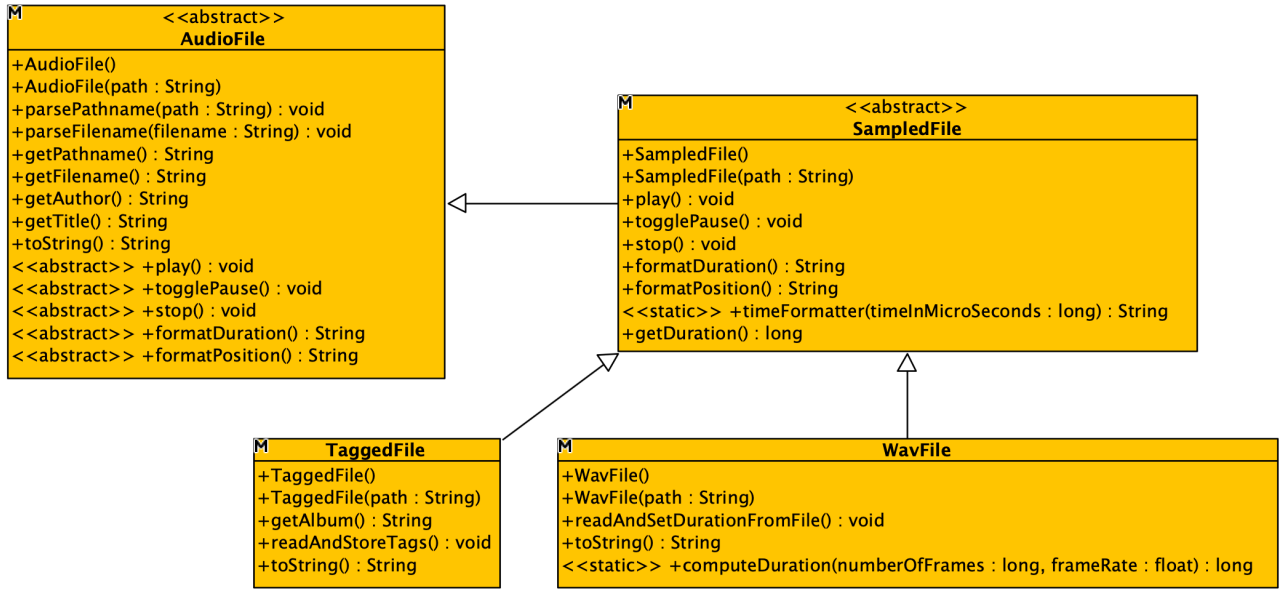
Kopieren Sie ihr Projekt zur vorangegangenen Vorführaufgabe und führen Sie folgende Aktionen durch

- Entfernen Sie die Tests der VA06 im „cert“-Ordner
- Entpacken Sie „audiofiles.zip“ in einen Ordner „audiofiles“ in ihrem Projekt.
- Entpacken Sie „lib.zip“ in einen Ordner „lib“ in ihrem Projekt.
- Fügen Sie dann **jede** dieser Jar-Bibliotheken dem Klassenpfad Ihres Projektes hinzu. In Eclipse finden Sie dazu im Kontextmenü des Projektes, dass Sie in der Standardansicht mit einem Rechtsklick auf das Projekt im Package-Explorer auf der linken Seite erreichen, den Eintrag "Build-Path“ → “Configure Build Path“ → “Libraries“ → “Add Jars“.

Achtung: Fügen Sie die Jar-Dateien dem „classpath“ hinzu, nicht dem „module path“!

Teilaufgabe a) Erzeugen der Klassenhierarchie

Das folgende UML-Klassendiagramm zeigt die Klassenhierarchie mit den Methoden der einzelnen Klassen.



Erweitern Sie die Klasse `AudioFile` und definieren Sie die fehlenden Klassen mit den benötigten Konstruktoren und Methoden so, dass diese ohne Fehler übersetzt werden können. Implementieren Sie in den Konstruktoren eine entsprechende Konstruktorenverkettung.

Hinweis: Bei Methoden mit einem Rückgabetyp ungleich „void“ genügt es, vorläufig „null“ oder „0“ zurückzugeben.

Lesetest für Klasse AudioFile

Beim Erzeugen einer `AudioFile`-Instanz soll nach dem Parsen des übergebenen Dateipfades

geprüft werden, ob der normalisierte Dateipfad auf eine lesbare Datei verweist. Ist das nicht der Fall, so soll eine sogenannte „RuntimeException“ geworfen werden, welche zum Abbruch des Programms führt.

Hinweise:

1. Für den Lesetest sollten Sie zuerst eine Instanz der Klasse „File“ erzeugen und die Instanz nutzen, um die Lesbarkeit („canRead“) zu prüfen. Nutzen Sie für die Details die Java Dokumentation z.B. hier: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/File.html
2. Eine Exception wirft man mit der Anweisung „throw“ nach folgendem Muster:

   ```
   throw new <Exceptiontyp>(<Fehlermeldung>);
   ```

   Eine Exception beschreibt eine Fehlersituation, Details dazu kommen im entsprechenden Kapitel der Vorlesung. Wenn sie geworfen wird, wird die aktuelle Methode sofort verlassen (kein regulärer Rücksprung zum Aufrufer).

Abspielen von Audiodateien in Klasse SampledFile

Jede Audiodatei soll wiedergegeben werden können. Die Wiedergabe soll darüber hinaus pausiert bzw. gestoppt werden können. Die Klasse `AudioFile` gibt diese Möglichkeiten über entsprechende Methoden abstrakt vor. In der Klasse `SampledFile` soll diese Schnittstelle nun mit Hilfe der Klasse studiplayer.basic.BasicPlayer implementiert werden:

- Zum Abspielen einer Audiodatei bietet `BasicPlayer` die Methode void `play(String path)`.
- Das Pausieren kann mit der Methode `void togglePause()` erreicht werden.
- Eine Wiedergabe wird mit `void stop()` angehalten.

Zum Testen der oben beschriebenen Schritte können Sie die bereitgestellten Tests in der **Datei TestSubtaskA.java** nutzen.

Teilaufgabe b) Einlesen der Tags für Tagged Files

Wie bereits beschrieben enthalten einige Audiodatei-Typen (z.B. mp3, ogg) Metadaten in Tag-Form. Ein Tag ist dabei eine Zeichenkette (z.B. „title“), zu der ein Wert gespeichert ist (z.B. „Rock me Amadeus“).

Die folgende Snippet zeigt alle in der Audiodatei „Rock 812.mp3“ gespeicherten Tags mit ihren Werten in der Konsole an (siehe auch **TestSubtaskB.java**):

```
TaggedFile tf = new TaggedFile("audiofiles/Rock 812.mp2");
Map<String, Object> tagMap = TagReader.readTags(tf.getPathname());
for (String tag : tagMap.keySet()) {
    Object value = tagMap.get(tag);
    System.out.printf("key: %-25s value: %-30s (type: %s)\n",
        key, value, value.getClass().getSimpleName());
}
```

Erläuterungen:

1. Ein Map ist eine Datenstruktur, die Schlüssel-Wert-Paare speichert. Durch den Aufruf des Tagreaders erhält man ein Map zu der angegebenen Audiodatei.
2. Durch Iteration über die Menge der Schlüssel (engl. *keyset*) erhält man jeweils ein Tag, zu welchem man mit *get(tag)* den dazugehörigen Wert ermitteln kann.
   Achtung: ist ein Schlüssel nicht vorhanden, so gibt get() den Wert „null“ zurück!
3. Beachten Sie bitte, dass die Werte unterschiedliche Datentypen haben. Dies sollten Sie in ihrem Programmcode entsprechend berücksichtigen.

Von den durch obigen Test ausgegebenen Tags interessieren uns „title“, „author“, „album“ und „duration“. Implementieren Sie nun die Methode

```
public void readAndStoreTags()
```

der Klasse `TaggedFile`. Die Methode soll die vier interessierenden Tags mit ihren Werten lesen und – falls ein Wert vorhanden ist – in entsprechenden Attributen speichern.

Stellen Sie darüberhinaus sicher, dass beim Erzeugen einer `TaggedFile`-Instanz das Einlesen der Tags angestossen wird.

Zum Testen der oben beschriebenen Schritte können Sie die bereitgestellten Tests in der **Datei TestTeilaufgabeB.java** nutzen.

Teilaufgabe c) Einlesen der Metainformation für WAVE-Dateien

Bei WAVE-Dateien muss die Gesamtspielzeit aus den in der Metainformation enthaltenen Werten für die Anzahl der Frames pro Sekunde (*framerate*) und der Anzahl der Frames (*numberOfFrames*) berechnet werden.

Dieser Vorgang soll beim Erzeugen einer `WavFile`-Instanz durch Aufruf der Methode `readAndSetDurationFromFile()` angestoßen werden. Folgende statische Methoden der Klasse `WavParamReader` stehen zur Implementierung der Methode zur Verfügung:

- `void readParams(String path)`
  Liest die Metainformation der Audiodatei zu "path" ein
- `float getFrameRate()`
  Liefert die zuvor gelesene Anzahl Frames pro Sekunde
- `long getNumberOfFrames()`
  Liefert die zuvor gelesene Anzahl Frames der Audiodatei

Zu beachten ist, dass `readParams()` vor dem Lesen der beiden Werte aufgerufen werden muss!

Die eigentliche Berechnung der Gesamtspielzeit soll in der statischen Methode `computeDuration()` durchgeführt werden. Das Ergebnis der Berechnung sollte in einem Attribut gespeichert werden (vgl. Getter-Methode `getDuration()`).

Zum Testen der oben beschriebenen Schritte können Sie die bereitgestellten Tests in der **Datei TestSubtaskC.java** nutzen.

Teilaufgabe d) Formatierung von Abspieldauer und aktueller Position

An der Benutzeroberfläche soll später (Vorführaufgabe 10) die Abspieldauer und die aktuelle Position einer Audiodatei im Format "<Minuten>:<Sekunden>" angezeigt werden. Hierfür sind die beiden Methoden `formatDuration()` und `formatPosition()` vorgesehen. Beide Methoden geben einen String zu einem Microsekunden-Wert zurück, weshalb die eigentliche Formatierung mit der statischen Methode `timeFormatter()` modularisiert werden soll.

String formatDuration()

Liefert die Gesamtspielzeit (vgl. `getDuration()`) im Format mm:ss

String formatPosition()

Liefert die aktuelle Abspielposition im Format mm:ss.

Hinweis:

Die Klasse `BasicPlayer` bietet hierfür die statische Methode `long getPosition()` an.

String timeFormatter(long timeInMicroSeconds)

Berechnet die Werte für Minuten und Sekunden aus dem übergebenen Mikrosekunden-Wert und liefert einen entsprechend formatierten String zurück.

Hinweise:

1. Zur einfachen Formatierung von Werten kann die statische Methode `format()` der Klasse String genutzt werden. Siehe Java-Dokumentation z.B. unter https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html
2. Ungültige Parameterwerte sollten zu RuntimeExceptions führen. Ungültige Werte sind negative Werte sowie Werte, die nicht in dem vorgegebenen „mm:ss“-Format dargestellt werden können.

Zum Testen der oben beschriebenen Schritte können Sie die bereitgestellten Tests in der **Datei TestSubtaskD.java** nutzen.

Teilaufgabe e) Restliche Methoden

Implementieren Sie zu guter Letzt die drei noch fehlenden Methoden:

String getAlbum()

Für `TaggedFile`-Instanzen soll diese Getter-Methode das beim Einlesen der Tag-Information gespeicherte Album liefern.

String toString()

Die bereits von `AudioFile` überschriebene `toString`-Methode soll in den beiden konkreten Klassen noch einmal überschrieben werden, um die zusätzlichen Informationen zurückgeben zu

können.

Für `WavFile`-Instanzen sollen die `AudioFile`-Stringrepräsentation und die formatierte Gesamtspielzeit durch „ – “ getrennt zurückgegeben werden.

Für `TaggedFile`-Instanzen gilt das für `WavFile`-Instanzen beschriebene Vorgehen, es sei denn, `album` hat einen „nicht-null“ Wert. Dann sollen die `AudioFile`-Stringrepräsentation, das Album und die formatierte Gesamtspielzeit durch „ - “ getrennt zurückgegeben werden.

Zum Testen der oben beschriebenen Schritte können Sie die bereitgestellten Tests in der **Datei TestSubtaskE.java** nutzen.

Hinweise zur Abnahme Ihrer Implementierung der Vorführaufgabe 07

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

Wenn alle Tests ohne Fehler durchlaufen, mailen Sie bitte folgende Java-Dateien als Dateianhang mit dem Betreff „VA07“ an den APA-Server:

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