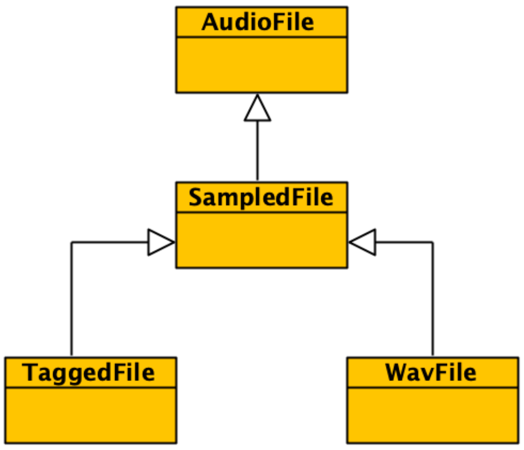
Task 7: Inheritance Hierarchy

The aim of this task is to implement the class hierarchy shown on the right providing the basic functionalities of the audio player.



The `AudioFile` base class has already been created in task 6. A class `SampledFile` is derived from it, which represents sampled audio files - such as MP3, OggVorbis or RIFF WAVE. The class uses classes contained in external Java archives (JAR files) to decode the audio files.

In our project, no classes other than `SampledFile` are derived from `AudioFile`. Strictly speaking, the two classes could therefore be combined into one. However, there are audio files, such as MIDI, which are not sampled and would therefore be more cleanly derived directly from AudioFile. For this reason, we make this separation.

The `TaggedFile` class derived from `SampledFile` represents audio files with meta data such as ID3 tags for MP3 or Vorbis comments for OggVorbis. The other class derived from `SampledFile` represents RIFF WAVE files that do not provide meta information in the form of tags. Such files are obtained, for example, when CDs are "ripped" with a corresponding tool.

Class AudioFile

- Provides methods for parsing the file name of the audio file.
- Specifies abstract methods for playing the audio file, pausing or stopping the playback of the audio file and for formatting the playback duration and playback position.

Class SampledFile

- Implements the abstract methods specified in AudioFile.

Class TaggedFile

- Reads file properties from the tags of the audio file.

Class WavFile

- Reads file properties from the meta information of the audio file.

Project Setup

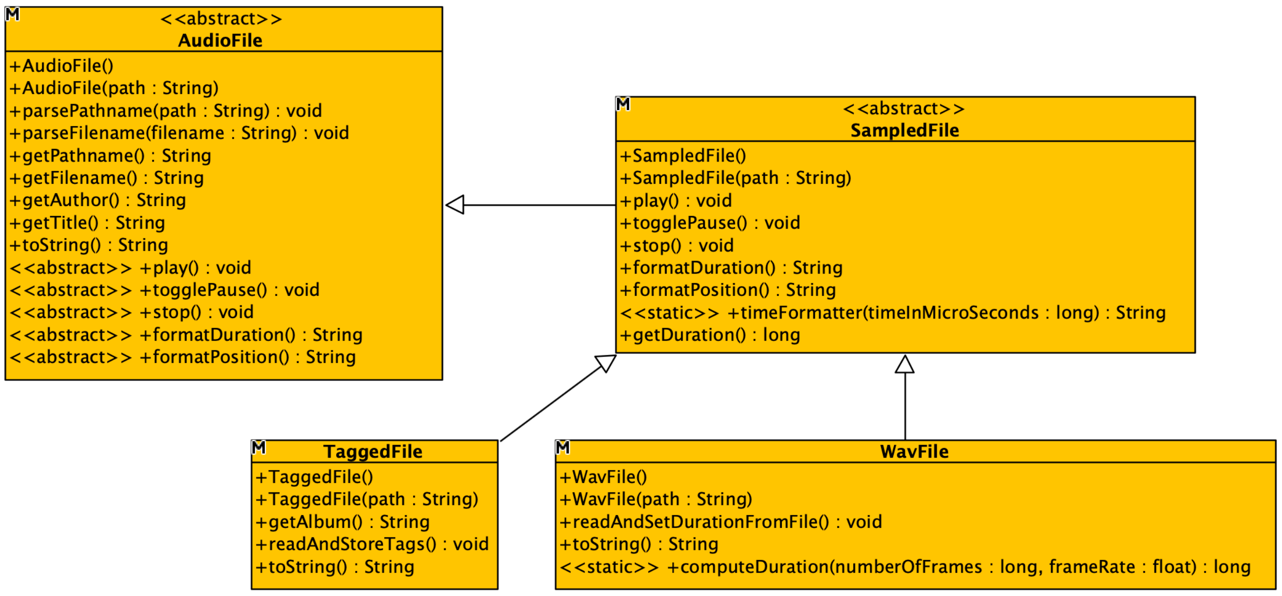
Copy your project from the previous task and carry out the following actions

- Remove all tests in the "cert" folder.
- Unzip "audiofiles.zip" into a folder "audiofiles" in your project.
- Unzip "lib.zip" into a folder "lib" in your project.
- Then add each of these jar libraries to the class path of your project. In Eclipse, you will find the entry "Build Path" → "Configure Build Path" → "Libraries" → "Add Jars" in the context menu of the project, which you can access in the standard view by right-clicking on the project in the package explorer on the left-hand side.

Important: Add the jar files to the "classpath", not the "module path"!

Subtask a) Create the class hierarchy

The following UML class diagram shows the class hierarchy including the methods of each class.



<<abstract>>
AudioFile
+AudioFile()
+AudioFile(path : String)
+parsePathname(path : String) : void
+parseFilename(filename : String) : void
+getPathname() : String
+getFilename() : String
+getAuthor() : String
+getTitle() : String
+toString() : String
<<abstract>> +play() : void
<<abstract>> +togglePause() : void
<<abstract>> +stop() : void
<<abstract>> +formatDuration() : String
<<abstract>> +formatPosition() : String
<<abstract>>
SampledFile
+SampledFile()
+SampledFile(path : String)
+play() : void
+togglePause() : void
+stop() : void
+formatDuration() : String
+formatPosition() : String
<<static>> +timeFormatter(timeInMicroSeconds : long) : String
+getDuration() : long
TaggedFile
+TaggedFile()
+TaggedFile(path : String)
+getAlbum() : String
+readAndStoreTags() : void
+toString() : String
WavFile
+WavFile()
+WavFile(path : String)
+readAndSetDurationFromFile() : void
+toString() : String
<<static>> +computeDuration(numberOfFrames : long, frameRate : float) : long

Extend the `AudioFile` class and define the missing classes with the required constructors and methods so that they can be translated without errors. Don't forget to implement appropriate constructor chainings!

Please note: For methods with a return type not equal to "void" it is sufficient to return "null" or "0" for the time being.

Reading test for class AudioFile

When creating an `AudioFile` instance, a check should be made after parsing the transferred file path whether the normalised file path refers to a readable file. If this is not the case, a so-called "RuntimeException" should be thrown, which leads to the programme being aborted.

Notes:

1. For the read test, you should first create an instance of the "File" class and use the instance to check the readability ("canRead"). Use the Java documentation for details, e.g. here: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/File.html
2. An exception is thrown with the "throw" instruction according to the following pattern:

   ```
   throw new <exception type>(<error message>);
   ```

   An exception describes an error situation, details on exception handling can be found in the corresponding chapter of the script. If an exception is thrown, the current method is exited immediately (no regular return to the caller).

Playing audio files in class SampledFile

It should be possible to play any audio file. It should also be possible to pause or stop playback. The `AudioFile` class provides abstract methods for this functionality which is now to be implemented in class `SampledFile` using class studiplayer.basic.BasicPlayer:

- To play an audio file, `BasicPlayer` offers the method `void play(String path)`.
- Pausing can be achieved with the method `void togglePause()`.
- Playback is stopped with `void stop()`.

To test the steps described above, you can use the tests provided in file **TestSubtaskA.java**.

Subtask b) Reading in tags of Tagged Files

As already described, some audio file types (e.g. mp3, ogg) contain metadata in tag form. A tag is a character string (e.g. "title") for which a value is stored (e.g. "Rock me Amadeus").

The following snippet displays all tags stored in the audio file "Rock 812.mp3" with their values in the console (see also **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());
}
```

Explanations:

1. A map is a data structure that stores key-value pairs. A map for an audio file is obtained by calling the tag reader with the respective filename as parameter.
2. When iterating over the set of keys, a tag is obtained on each iteration for which the corresponding value can be determined by calling *get(tag)*.
3. Attention: if a key does not exist, *get()* returns the value "null"!
4. Please note that the values have different data types. You should take this into account

accordingly in your program code.

Of the tags output by the above test, we are only interested in "title", "author", "album" and "duration". Now implement the method

```
public void readAndStoreTags()
```

of class `TaggedFile`. The method should read the four tags with their values and - if a value is available - save them in corresponding attributes.

In addition, make sure that the reading of the tags is triggered when a `TaggedFile` instance is created.

To test the steps described above, you can use the tests provided in file **TestSubtaskB.java**.

Subtask c) Reading in metainformation for WAVE files

For WAVE files, the total playing time must be calculated from the values contained in the meta information for the number of frames per second (frame rate) and the number of frames.

This process is to be triggered when a `WavFile` instance is created by calling the method `readAndSetDurationFromFile()`. The following static methods of the `WavParamReader` class are available for implementing the method:

- `void readParams(String path)`
  Reads the meta information of the audio file for "path"
- `float getFrameRate()`
  Returns the previously read number of frames per second
- `long getNumberOfFrames()`
  Returns the previously read number of frames of the audio file

Please note that `readParams()` must be called before reading the two values!

The actual calculation of the total playing time should be carried out in the static method `computeDuration()`. The result of the calculation should be saved in a corresponding attribute (see getter method getDuration()).

To test the steps described above, you can use the tests provided in **TestSubtaskC.java**.

Subtask d) Formatting duration and current position

The playback duration and the current position of the selected audio file are to be displayed in the format "<minutes>:<seconds>" on the user interface (see task 10). The two methods `formatDuration()` and `formatPosition()` are provided for this purpose. Both methods return a string from a microsecond value, which is why the formatting should be modularised with the static method `timeFormatter()`.

String formatDuration()

Returns the total playing time (see `getDuration()`) in the format mm:ss

String formatPosition()

Returns the current position in the format mm:ss

Note:

The `BasicPlayer` class offers the static method `long getPosition()` for this purpose.

String timeFormatter(long timeInMicroSeconds)

Calculates the values for minutes and seconds from the given microsecond value and returns a correspondingly formatted string.

Notes:

1. Use the static method `format()` of class String to format values in a string. See the Java documentation e.g. here: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html
2. Invalid parameter values should lead to RuntimeExceptions. Invalid values are negative values and values that cannot be displayed in the specified "mm:ss" format.

To test the steps described above, you can use the tests provided in file **TestTSubtaskD.java**.

Subtask e) Remaining methods

Finally, implement the three missing methods:

String getAlbum()

For `TaggedFile` instances, this getter method should return the album saved when the tag information was read in.

String toString()

The `toString` method already overwritten by `AudioFile` should be overwritten again in the two concrete classes in order to be able to return the additional information.

For `WavFile` instances, the `AudioFile` string representation and the formatted total playing time should be returned separated by " - ".

For `TaggedFile` instances, the procedure described for `WavFile` instances applies, unless album has a "non-zero" value. In that case the `AudioFile` string representation, the album and the formatted total playing time should be returned separated by " - ".

To test the steps described above, you can use the tests provided in file **TestSubtaskE.java**.

Hints on Acceptance of Your Solution

Download all acceptance tests for task 7 and save them in the source folder “cert” of your project. Then run the acceptance tests in Eclipse.

If all tests run without errors, please email the following Java files as file attachments with the subject "ex-07" to the APA server:

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