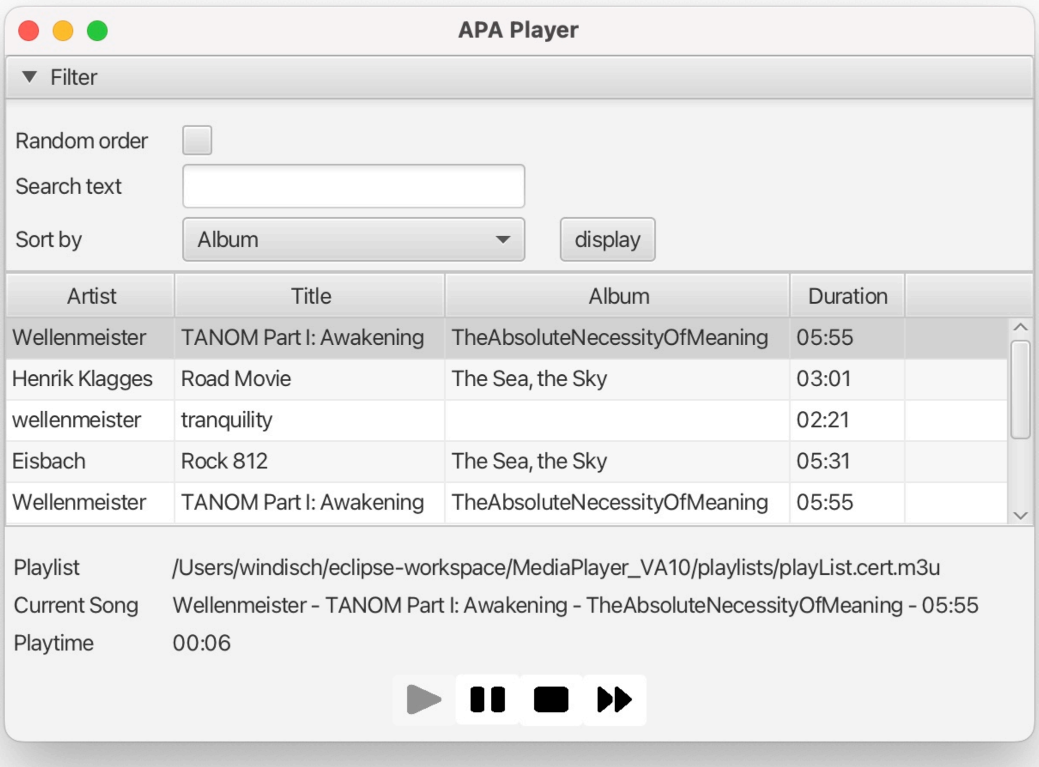
Task 6: Splitting File Paths

The aim of this practical course is to develop a media player for playing audio files. The following illustration shows the final user interface (UI) of the application, which consists of the player with a searchable and sortable play list and the control buttons.



Random order
Search text
Sort by: Album

Artist	Title	Album	Duration
Wellenmeister	TANOM Part I: Awakening	TheAbsoluteNecessityOfMeaning	05:55
Henrik Klagges	Road Movie	The Sea, the Sky	03:01
wellenmeister	tranquility		02:21
Eisbach	Rock 812	The Sea, the Sky	05:31
Wellenmeister	TANOM Part I: Awakening	TheAbsoluteNecessityOfMeaning	05:55

Playlist /Users/windisch/eclipse-workspace/MediaPlayer_VA10/playlists/playList.cert.m3u
Current Song Wellenmeister - TANOM Part I: Awakening - TheAbsoluteNecessityOfMeaning - 05:55
Playtime 00:06

Setup your development environment

Create a new project in your IDE, e.g. "MediaPlayer_Task6".

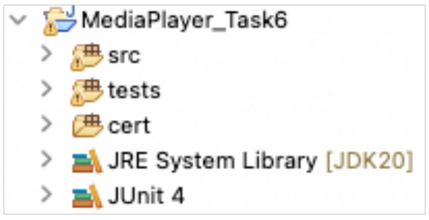
Note: It is assumed that you will be using Eclipse for Java as your IDE (Integrated Development Environment, see https://www.eclipse.org/downloads/). Of course, you are free to choose your development environment, but you will only get the best support for Eclipse during this practical course.

Attention: do not create a module info! (in Eclipse this is an option in the first window of the wizard)

Use suitable settings in your IDE to ensure that:

- a source folder[1] `src` is available for storing the Java sources
- a source folder[2] `test` is available for storing your own JUnit tests
- a source folder `cert` is available to store the acceptance tests
- JUnit version 4 is integrated as a dependency for tests in the project[3]

Your project should now look like this in Eclipse:



Basic information about paths and file names

The audio player created during the practical course can process audio data in the form of audio files that can be accessed via a file system. We therefore begin the implementation of the audio player by creating the `AudioFile` class.

Each object of the `AudioFile` class refers to a single audio file. The path name of this file consists of the optional path and the file name. A path can be an absolute or relative. This means that the full path, a relative path or no path at all can be specified.

Note: The character '‿' denotes a space.

Note: Please note that the path separator, i.e. the separator for the individual directories, differs depending on the operating system. Under Unix/MacOS it is the character '/', under Windows the character '\'.

Attention: To use the character '\' in Java, '\\' must be written, as '\' is a control character.

Examples for Unix/Linux/Mac systems:

/home/meier/Musik/Falco‿–‿Rock‿Me‿Amadeus.mp3	(absolute path, begins with /)
../musik/Falco‿–‿Rock‿Me‿Amadeus.mp3	(relative path)
Falco‿–‿Rock‿Me‿Amadeus.mp3	(no path)

[1] In Eclipse this folder will be present after creating the project

[2] In Eclipse: right click on project → New → Source Folder; Java sources contained in a source folder are automatically compiled.

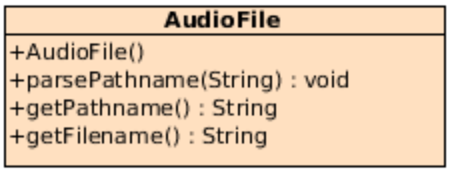
[3] In Eclipse: right click on project → Build Path → Add Libraries → JUnit → JUnit library version: JUnit 4

Examples for Windows systems:

D:\Daten\Musik\Falco‿–‿Rock‿Me‿Amadeus.mp	(absolute path, begins with a drive letter)
..\Musik\Falco‿–‿Rock‿Me‿Amadeus.mp3	(relative path)
Falco‿–‿Rock‿Me‿Amadeus.mp3	(no path)

Subtask a) Creating the AudioFile class

Create class `AudioFile` that provides the four methods shown in the class diagram.



The task of the `AudioFile` class is the decomposition and operating system-dependent normalisation of a given path name. The results are to be stored in attributes that can be read outside the `AudioFile` object using getter methods (e.g. getPathname()).

Note: A method `getPathname()` usually points to an attribute `pathname`, analogously this applies to the method `getFilename()` and others.

Subtask b) Parsing paths

Implement the two parse methods according to the following specifications.

```
parsePathname(String path): void
```

The `path` parameter contains the path name of the audio file. The method should analyse the parameter and assign values to the attributes `pathname` and `filename` as a result of the analysis. In detail, the following requirements should be fulfilled, examples can be found in the table:

- **Handle drive letter**
  If the programme does not run under Windows, a drive specification at the beginning (e.g. "C:\...") should be corrected.
- **Normalize path separators**
  Path separators contained in `path` should be converted into the separators valid for the current operating system. In addition, sequences of path separators (e.g. "medien///song.mp3") should be combined into one (e.g. "medien/song.mp3").
- **Savin the results in attributes**
  The result of the above operations should be saved in `pathname`. If there are path separators in path, the value for `filename` results from the character string to the right of the last path separator. Otherwise, `filename` will have the same value as `pathname`.

The following table contains examples that illustrate and supplement the above requirements.

Argument for parsePathname()	Result of getPathname()	Result of getFilename()
Empty string or only spaces or tabs	Empty string	Empty string
file.mp3	file.mp3	file.mp3
‿‿/my-tmp/file.mp3	Linux: ‿‿/my-tmp/file.mp3 Windows: ‿‿\my-tmp\file.mp3	file.mp3
//my-tmp////part1//file.mp3/	Linux: /my-tmp/part1/file.mp3/ Windows: \my-tmp\part1\file.mp3\	Empty string
d:\\\\part1///file.mp3	Linux: **/d/**part1/file.mp3 Windows: **d:**\part1\file.mp3	file.mp3
-	-	-
‿-‿	-	-

To develop the `parsePathname()` method, you can use the **test bed provided in the TestParsePathname.java**. The tests check the examples listed in the table and a few more.

Testing the class under Windows and Linux

To ensure that your application runs correctly on Windows and Linux, we use the system properties "os.name" (operating system name) and "file.separator" (path separator) both in the `AudioFile` class and in the JUnit tests. The properties are set by Java before the test is executed. The required lines of code are already available, please familiarise yourself with the tests and run them for both operating systems.

In some places it is also important to know whether the programme is executed under Windows. You can use the following auxiliary method for this:

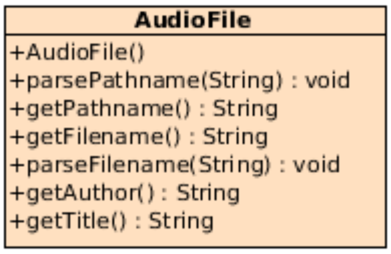
```
private boolean isWindows() {
    return System.getProperty("os.name").toLowerCase()
        .indexOf("win") >= 0;
}
```

Subtask c) Parsing file names

In this subtask, the method `parseFilename(String filename)` is to be developed to analyse a given file name. Audio file names are often structured according to the following scheme:

```
Author‿-‿Title.Extension
```

Author indicates who wrote the song with the title, extension typically takes the values "mp3", "mp4", "wav" etc.



`parseFilename(String filename)` should split the transferred file name into the two components mentioned above and save the result of the splitting in the new attributes `author` and `title`. Getters for `author` and `title` should enable access to the analysed values. The UML class diagram on the right shows the extended `AudioFile` class.

The following table uses examples to show how the new parse method should work.

Argument for parseFilename()	Result of getAuthor()	Result of getTitle()
‿Falco‿‿-‿‿Rock‿me‿‿‿‿Amadeus‿.mp3‿‿	Falco	Rock‿me‿‿‿‿Amadeus
Frankie‿Goes‿To‿Hollywood‿-‿The‿Power‿Of‿Love.ogg	Frankie‿Goes‿To‿Hollywood	The‿Power‿Of‿Love
audiofile.aux	Empty string	audiofile
‿‿‿A.U.T.O.R‿‿‿-‿‿T.I.T.E.L‿‿.EXTENSION	A.U.T.O.R	T.I.T.E.L
Hans-Georg‿Sonstwas‿-‿Blue-eyed‿boy-friend.mp3	Hans-Georg‿Sonstwas	Blue-eyed‿boy-friend
.mp3	Empty string	Empty string
Falco‿-‿Rock‿me‿Amadeus.	Falco	Rock‿me‿Amadeus
-	Empty string	-

To develop the new methods, you can use the **tests in TestParseFilename.java**.

Subtask d) Main Constructor for AudioFile

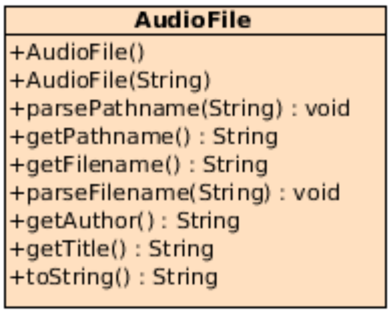
Previously, the parse methods were called on an object created with the default constructor. However, the aim is to specify the path and perform the analyses when an AudioFile object is created. As a result, all four attributes (`pathname`, `filename`, `author`, `title`) are assigned after creation.

Implement a parameterised constructor `AudioFile(String path)` that expects the path name of an audio file as an argument; this will be the main constructor. The default constructor should still be available.

Subtask e) String representation for AudioFile objects

For the output of objects of the `AudioFile` class, we overwrite the `toString()` method in the AudioFile class. The method should return a string according to the following specifications:

- If `getAuthor()` returns an empty string, only the title should be returned
- Otherwise, the method should return the artist and title separated by "‿-‿"



To develop the `toString` method, you can use the **tests in TestToString.java**. Please run these tests for <u>both operating systems</u> (see method `setUp()`).

Implementing the `toString` method results in the class diagram shown on the right side.

Hints on acceptance of your solution

Download all acceptance tests belonging to the task sheet and save them in the source folder **cert** of your project (see section "Preparation" at the beginning of this task sheet). Then run the acceptance tests in Eclipse.

Attention: test your solution with both operating systems by setting the corresponding line as a comment line in the setUp method of AudioFileTest.java:

```
// sep = Utils.emulateWindows();
```

…and activate linux emulation:

```
sep = Utils.emulateLinux();
```

If all tests run without errors for both operating systems, please send file AudioFile.java as an attachment to apa@thi.de with subject “ex-06”.