# *MP3Tag Editor  :  Program overview*

**Authors:** Florian Schultze, Sebastian Wjertzoch, Paweł Kłeczek

## Introduction

Our MP3Tag editor was creatid using Model-View-Controler design pattern.

The *Model* contains only data structures used to store information about MP3 files, similarly the *View* consist only of controls-related code. The „heart” of our program is the *Control* part, which provides the main functionality.

## Model

When implementing a tree structure we have decided not to base on `DefaultTreeModel` and `MutableTreeNode`, but to create our own implementations. This approach was more convenient when it came to the implementation of XML cache.

The tree structure consist of two node types: `MP3Folder` and `MP3File` (which both have the same parent, `MP3Object`).

The `MP3TagFrame` is also the child of `MP3Object.` This solution also helps us to write XML data more easily.

## View

The *View* part is covered by `Application` class. To achieve qualified GUI for our application (controls and layout), the built-in GUI builder of NetBeans IDE was used.

All changes of the GUI are reported using Observer-Observable design pattern (*View* observes events from *Control* and other event-generating modules).

Controls register their own listeners, which call the same functions (eg. each text field calls the same `textFieldKeyReleased()` method). Than it is decided from which control an event comes and handed over to the *Control*.

ID3 tag specification states that a valid year consist of 4 digits. In our program it's verification occurs when the year text field looses it's focus. It's content is then compared with the pattern and if the test fails, the previous (correct) date is restored. Blank year is also a valid value, it indicates that there is no such a frame (or that the year frame should be removed).

## Control

### *XML reading / writing*

As the data which our program processes can be described as „object-oriented”, it makes sens to make use of this. That is why we have chosen JAXB as our XML mechanism provider (it simply maps Java class to an XML representation).

The guide on JAXB can be found here: http://jaxb.java.net/guide/

To configure JAXB the developer must add some annotations (eg. `@XmlRootElement` or `@XmlElements`), which instructs JAXB how to deal with the certain object: which fields should be mapped, which name wo use for each mapped field etc. Then it takes only a few lines of code to read or write the entire data structure (see `XMLReader` and `XMLWriter` classes).

To improve our program's performace we first check whether a cache of the chosen folder already exists. If so, then we make sure it contains actual data by comparing modification dates of both objects.

### *Threads*

In order to make our program running more smoothly and to give the user an opportunity to still modify the data, even when the saving process is running, a complete „deep-copy” of the whole data structure is passed to the writer thread. This is achieved by implementing `Cloneable` interface in all persisten classes and using `clone()` method for copying.

Because of that and because „jobs” are stored in a queue which is processed in a *first-in-first-out* order there is no need to incorporate any kind of synchronization mechanisms in our solution.

## Features

It is possible to add folders to the tree using drag-and-drop method. Simply drag a folder from system's file explorer and drop it on the tree panel!

Modified files are clearly visible by marking it in the tree structure with a *(*)* symbol.

The *Ordneransicht* tab provides the user with a brief summary of the content of currently selected folder or file.

When the user closes the editor and there are still some files waiting to be saved, a special warning dialog is shown. It is then possible either to exit instantly or to wait until the writing process is finished (the editor then closes automatically). This mechanism prevents user from loosing important information due to accidental closure of the program.