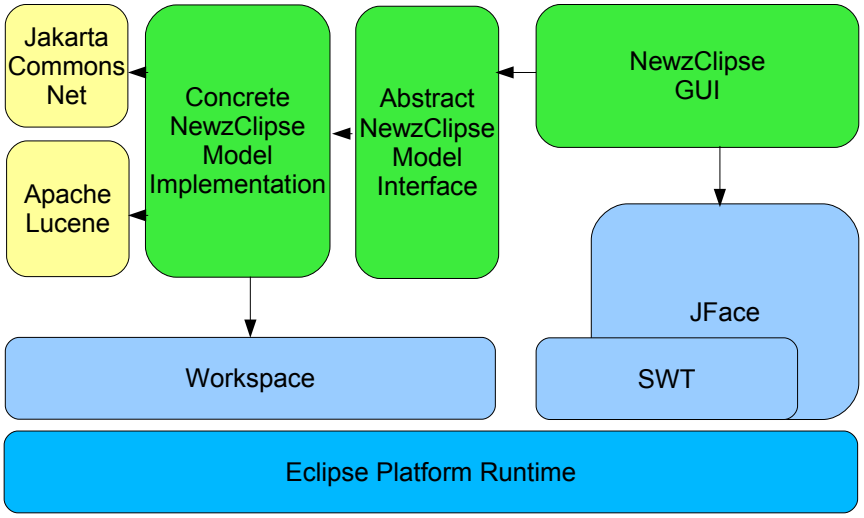
# 4 High-Level Design

The NewzClipse Project is split into two plugins: one for the model and one for the GUI[1]. The model plugin provides abstract classes that allow the GUI to access newsservers, newsgroups and articles and search through them. A factory class is also provided that allows the GUI plugin to instantiate concrete objects where neccessary. The fact that the information underlying those abstract objects are load over NNTP from the network or from some local disk space is hidden from the GUI. This is implemented in subclasses that are not visible to the user interface plugin. The model plugin also contains the two external libraries Apache Commons Net and Apache Lucene.

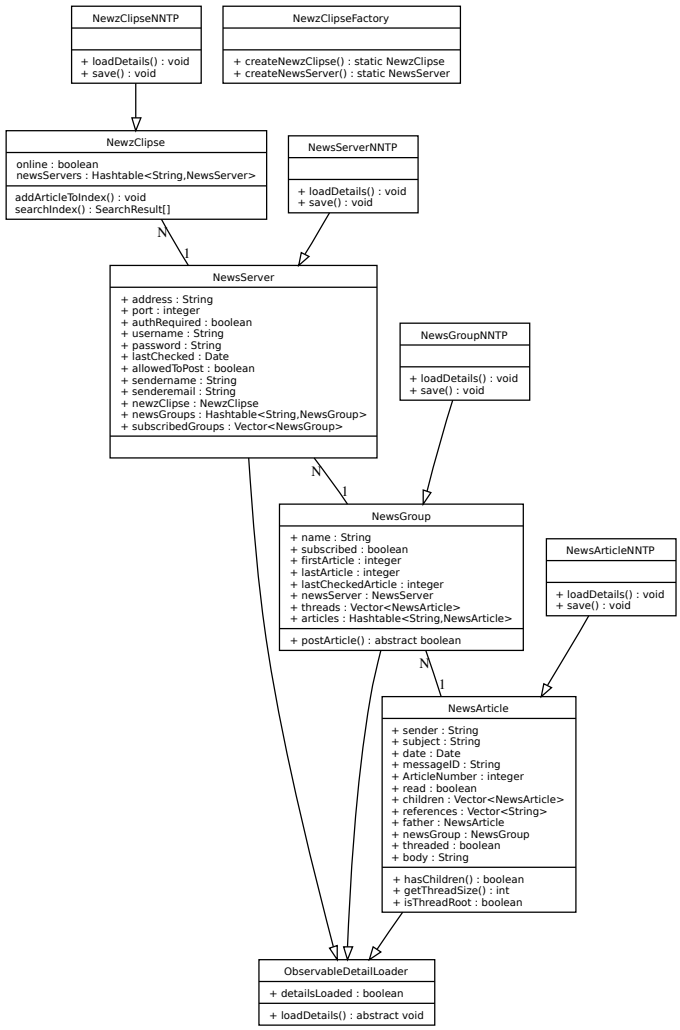



The following chapters will discuss some details of the implementation. Chapter 5 will deal with the model, its abstract base classes as well as the concrete implementation. Chapter 6 is about searching and the decision to use Apache Lucene. And finally Chapter 7 is about the GUI plugin.

[1] GUI: Graphical User Interface

# 5 Model

## 5.1 UML Overview



## 5.2 Abstract Model Interface

The preceeding UML diagram shows that Usenet servers (class `NewsServer`) contain subscribed and unsubscribed newsgroups (class `NewsGroup`). These newsgroups contain threads which are basically nothing else but articles in a newsgroup without a father. Therefore both, articles and threads, are stored in an object of class `NewsArticle`.

NewzClipse allows the user to access several NNTP servers. These are stored in a containing father object of class `NewzClipse`. These classes are all abstract as they only provide an interface to the model but require concrete subclasses to provide functions that load the objects from the network or the disk and maintain a search index.
The GUI code only works with this abstract classes and, if neccessary, instantiates new objects with the class `NewzClipseFactory` and its static functions. This class will then automatically deliver concrete implementations.

One principle that holds for `NewsServer`, `NewsGroup` and `NewsArticle` is that of lazy loading. Some basic information is loaded at the moment when such an object is created but more detailed information is only brought to memory if it is requested. For example a new `NewsGroup` object will contain the name of the newsgroup and the number of the last loaded article. If the threads and articles of this object are accessed for the first time they first have to be loaded.
This has the advantage that not all newsgroups and articles have to be in memory as soon as a `NewzClipse` object is created. This would be very memory consumptive and inefficient. This functionality is defined in an abstract way in the class `ObservableDetailLoader`, which also extends the Java standard class `Observable` because it will be needed in subsequent classes and Java does not provide the possibility of multiple inheritance.
The next section will discuss how the concrete implementation of this model work.

## 5.3 Concrete Model Implementation

### 5.3.1 Choice of Technology

The concrete implementation of the abstract model interface discussed in the previous section consists of the classes `NewzClipseNNTP`, `NewsServerNNTP`, `NewsGroupNNTP` and `NewsArticleNNTP`. It is based on NNTP to retrieve the news from the Usenet and on XML for local persistency. NNTP has been chosen as the most convenient method and standard for accessing newsgroups. XML was used as it can be used easily in Java and leaves room for further extensions.

At this level, the implementational details are hided from the upper layers. One could equally write another implementation that could be used by the GUI without a change, that would access the Usenet over the Google Groups web interface and store the data

locally in a relational database.

### 5.3.2 The Load Cycle - From the GUI to an Article

The GUI will instantiate a `NewzClipseNNTP` object transparently via the static method `NewzClipseFactory.createNewzClipse(String basePath)`. This object will check if it is able to find an appropriate directory containing the NewzClipse data below the specified parameter `basePath`. If this is not the case it will be created.
In this directory the top level XML file is read (if it exists or otherwise created) that contains the data about all news servers specified by the user. The corresponding `NewsServerNNTP` objects are also created and added to the just created `NewzClipseNNTP` object. They only contain the basic information from the XML file.

Now, let us suppose the user wants to see a list of all subscribed groups of one of the servers. The `NewsServerNNTP` will check if this information is available offline on the hard disk in an XML file that is maintained in a seperate subdirectory for each news server.
If this is not the case it will request a `NNTPClient` (part of the Jakarta Commons Net library) from its self-maintained connection pool and loads the list of newsgroups from the network. Now, if a list of articles of this group is requested it is again first checked if this was already downloaded and thus available offline. The same happens if the user finally requests a `NewsArticle`. All access to the NNTP servers happen over a connection pool maintained in each `NewsServerNNTP` object. The contents and structure of the files are described in the next section.

### 5.3.3 XML Formats

The directory that contains this file also contains a directory for each newsserver which in turn contains the NewsServer file, described afterwards.

#### NewzClipse Top Level File

Top-Level Element: <NewzClipse>
For each `NewsServer` there is a node containing its basic data:

```
<NewsServer>
   <address>
      <!-- IP Address or DNS domain -->
   </address>
   <port>
      <!-- TCP Port of the server -->
   </port>
   <authRequired>
      <!-- true/false, if authentication is/is not required -->
   </authRequired>
```

```
    <username>
        <!-- Username for authentication -->
    </username>
    <password>
        <!-- corresponding password -->
    </password>
    <allowedToPost>
        <!-- true/false, whether posting is allowed or not -->
    </allowedToPost>
    <sendername>
        <!-- Name that is be displayed when articles are written -->
    </sendername>
    <senderemail>
        <!-- which email should be used for sending articles -->
    </senderemail>
    <lastChecked>
        <!-- Date: when was last checked for new newsgroups -->
    </lastChecked>
</NewsServer>
```

#### NewsServer File

Top Level Element: <NewsServer>
For each NewsGroup a child element like this:

```
<NewsGroup>
    <name>
        <!-- Newsgroup Name -->
    </name>
    <firstArticle>
        <!-- first article number that was loaded -->
    </firstArticle>
    <lastArticle>
        <!-- last article that is available from this group -->
    </lastArticle>
    <lastCheckedArticle>
        <!-- last article that was loaded -->
    </lastCheckedArticle>
    <subscribed>
        <!-- true/false, if group is/is not subscribed by user -->
    </subscribed>
</NewsGroup>
```

The folder with this folder again contains subfolders, one for each subscribed newsgroup, containing the following file.

### NewsGroup File

Top Level Element: <NewsGroup>
Child elements for every contained NewsArticle:

```
<NewsArticle>
   <sender>
      <!-- Author of the message -->
   </sender>
   <read>
      <!-- true/false, if user read/ did not read the article -->
   </read>
   <subject>
      <!-- Subject of message -->
   </subject>
   <messageID>
      <!-- unique identifier of article -->
   </messageID>
   <articlenumber>
      <!-- Identifier of article within newsgroup -->
   </articlenumber>
   <date>
      <!-- Date when article was sent -->
   </date>
</NewsArticle>
```

In this containing folder there is again another directory named messages that contains the body as returned from the newsserver, dumped into a file and named after the article number of the message.