Contents

[1. Context 2](#_Toc364255118)

[2. Introduction 3](#_Toc364255119)

[3. Index components 4](#_Toc364255120)

[Index entry points 4](#_Toc364255121)

[IndexSynchronizer 4](#_Toc364255122)

[Index task 5](#_Toc364255123)

[Content extractor 5](#_Toc364255124)

[Transient index 5](#_Toc364255125)

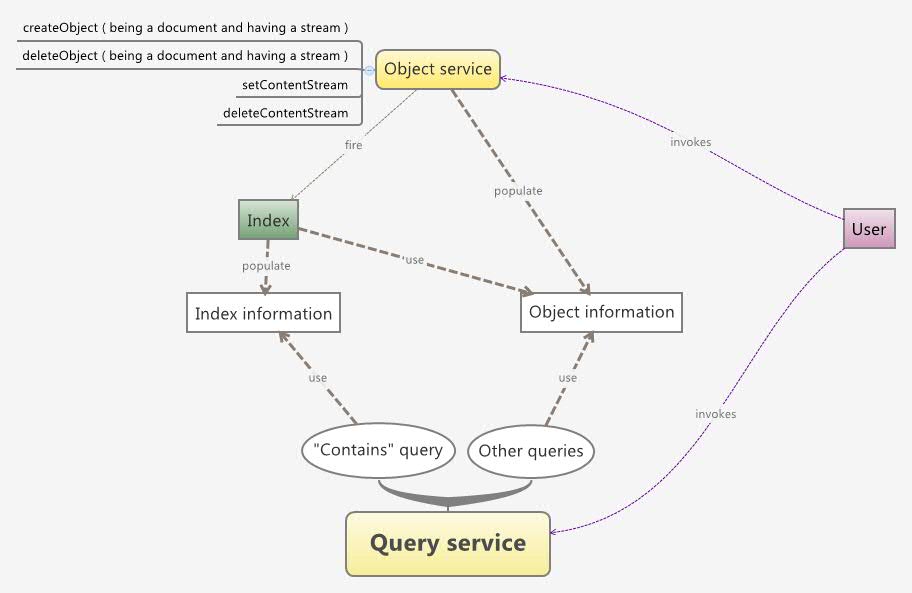
[Permanent index (Reverse index) 5](#_Toc364255126)

[4. The indexing process 6](#_Toc364255127)

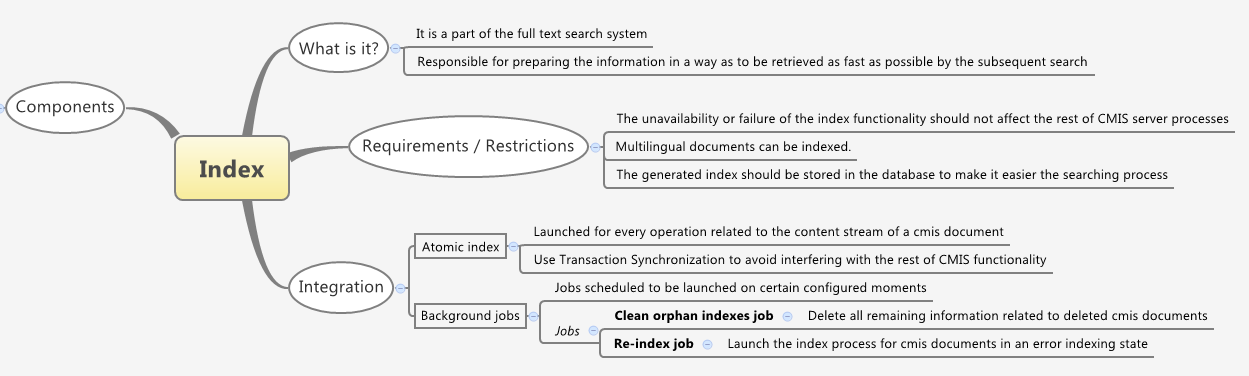
[5. Class Diagrams 7](#_Toc364255128)

# Context

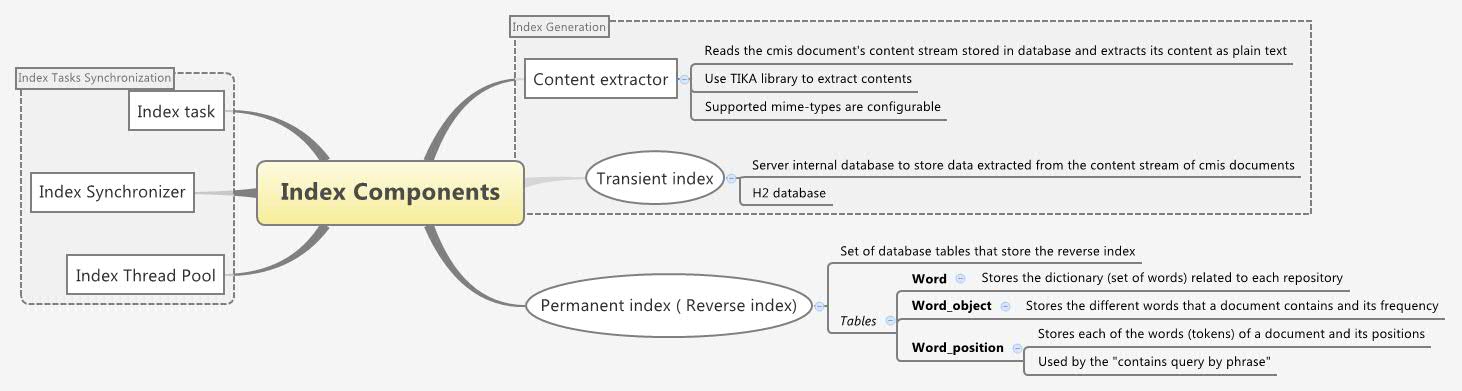
The index is an internal CMIS Server functionality. It is a piece of software at the service of the CMIS query service and it is responsible for populating the data structure needed by the query service to provide full text search in documents.



# Introduction



# Index components



## Index entry points

The entry points to the index functionality are the subclasses of AbstractIndex: IndexImpl and IndexCleanerJob.

IndexImpl is invoked by the CMIS object service to perform the indexing of single documents responding to user requests to the CMIS server (atomic index).

IndexCleanerJob is a programmed process that look for orphan or unfinished indexes and perform the convenient index operation for each of them.

The index entry points create IndexTasks with the needed information and pass them to the task executor (by means of AbstractIndex.executeIfAllowed() method), which in turn will use a ThreadPool to execute the operations specified by IndexTasks. To coordinate the case of having multiple IndexTasks related to the same cmis object, the AbstractIndex will use the IndexSynchronizer (thread save).

## IndexSynchronizer

The IndexSynchronizer is only a store for IndexTasks. It manages two maps:

- "executing map": the map that registers the IndexTasks that can be added to the task executor

- "waiting map": it is used to know if there is some other IndexTask (apart from executing one) programmed for a CMIS object so the executing task has to stop. Only the last arrived IndexTask is saved in this map.

## Index task

The index task is the object that manages all the indexing process of a single document (through the IndexHelper). It stores information about the CMIS object to index and the index operation to be performed (index or delete index).

Once the index task is chosen to be executed, it does the following job:

[Operation.INDEX]

1. Clean all the index information about that CMIS object from the *permanent index.*
2. Updates the CMIS object indexing state to NONE.
3. Get the object stream from database and extracts its contents using the *content extractor,* and save the information into the *transient index*.
4. Query the transient index grouping the information in different ways depending on the permanent index table to be populated and populate the *permanent index*.
5. Removes all the indexing information from the transient index.
6. [This step is performed always] Updates the CMIS object indexing state:
   1. The mime-type of the stream is not supported: NOT\_INDEXABLE
   2. The process was successful: INDEXED
   3. The process fail: ERROR

[Operation.INDEX\_DELETE]

1. Clean all the index information about that CMIS object from the *permanent index.*
2. Updates the CMIS object indexing state to NONE.

## Content extractor

Given a CMIS object identifier, the content's extractor constructs pipes that connect the stream related to that object in database with the Tika's component that is able to interpret it and read it word by word .

The whole stream is not loaded into memory but it loads chunks of the stream on demand.

## 

## Transient index

It is an H2 database embedded in the server to store the temporal information resulting from the content extraction process. This database consists of a sigle table: transient\_index.

## Permanent index (Reverse index)

It is the set of the external database tables used to store the reverse index information. It consist of three tables:

* Index\_word: stores the dictionary of a repository, meaning the different words of all the documents of a repository. The words are processed by a Lucene Analizer (StandardAnalizer) without stop words. So the words are all in lower case and without accents, punctuation marks..
* Index\_word\_object: stores the different words of a document with their frequency
* Index\_word\_position: stores all the words of the document with their order of appearance.

# The indexing process

The process is as follows:

* The entry point create an IndexTask, specifying which index operation (index or delete index) it is to be performed and some other data about the cmis object.
* The entry point use the IndexSynchronizer.putInQueue(IndexTask, waiting) method to check if it is possible to add the IndexTask to task executor.
  + If it is possible, it adds the index to the task executor, so the IndexTask.doIndex() will be executed.
  + If not, depending on the "waiting" argument, different things could happen:
    - waiting = false: the index synchronizer does nothing with the task and neither the entry point, so the IndexTask is lost. It is the way the background jobs act: if the IndexTask cannot be executed directly, it is ignored. This is to avoid stopping an executing atomic task.
    - waiting = true: the index synchronizer put the IndexTask in the "waiting map".
* The IndexTask is executed. When it finishes, it invokes the IndexSynchronizer.doOnTaskFinished() method.
* The IndexSynchronizer removes the IndexTask from the "executing map" and fires and IndexEvent
* The IndexImpl listen to that event, retrieves the waiting task for that cmis object from the IndexSynchronizer and begins the process with this other task.

# Class Diagrams

Figure 1. Index entry points.

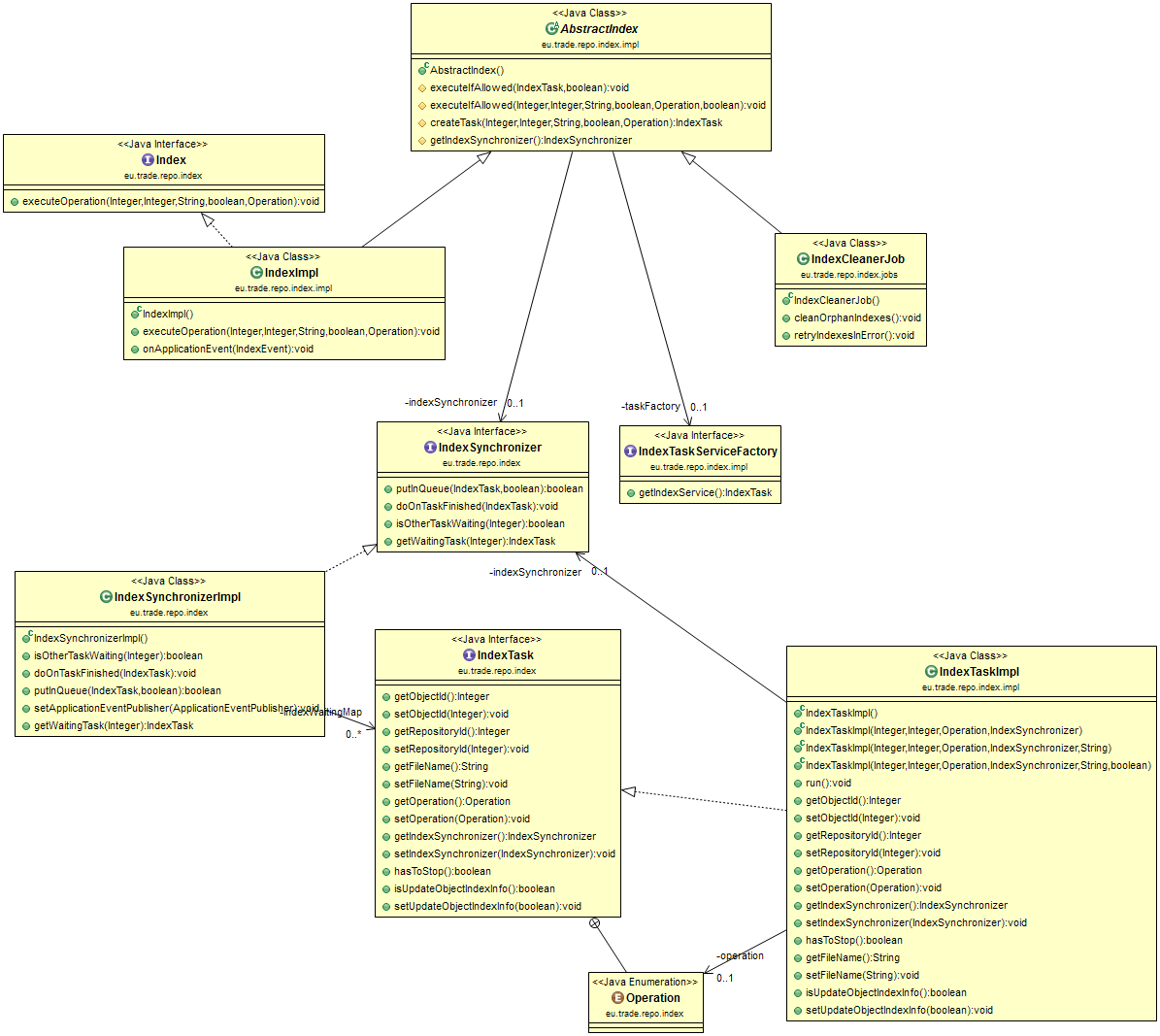


Figure 2. IndexTask

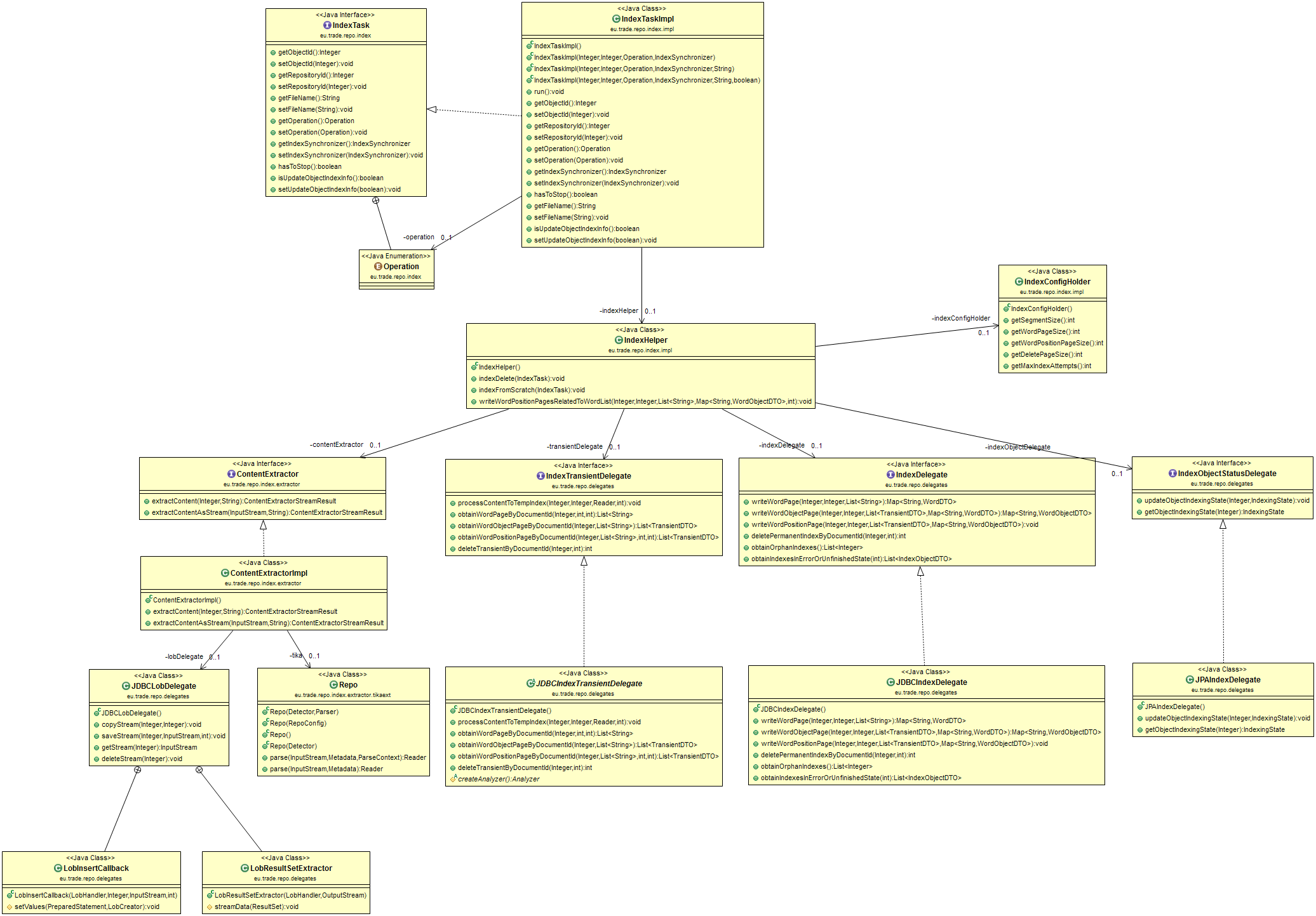


Figure 3. Atomic index

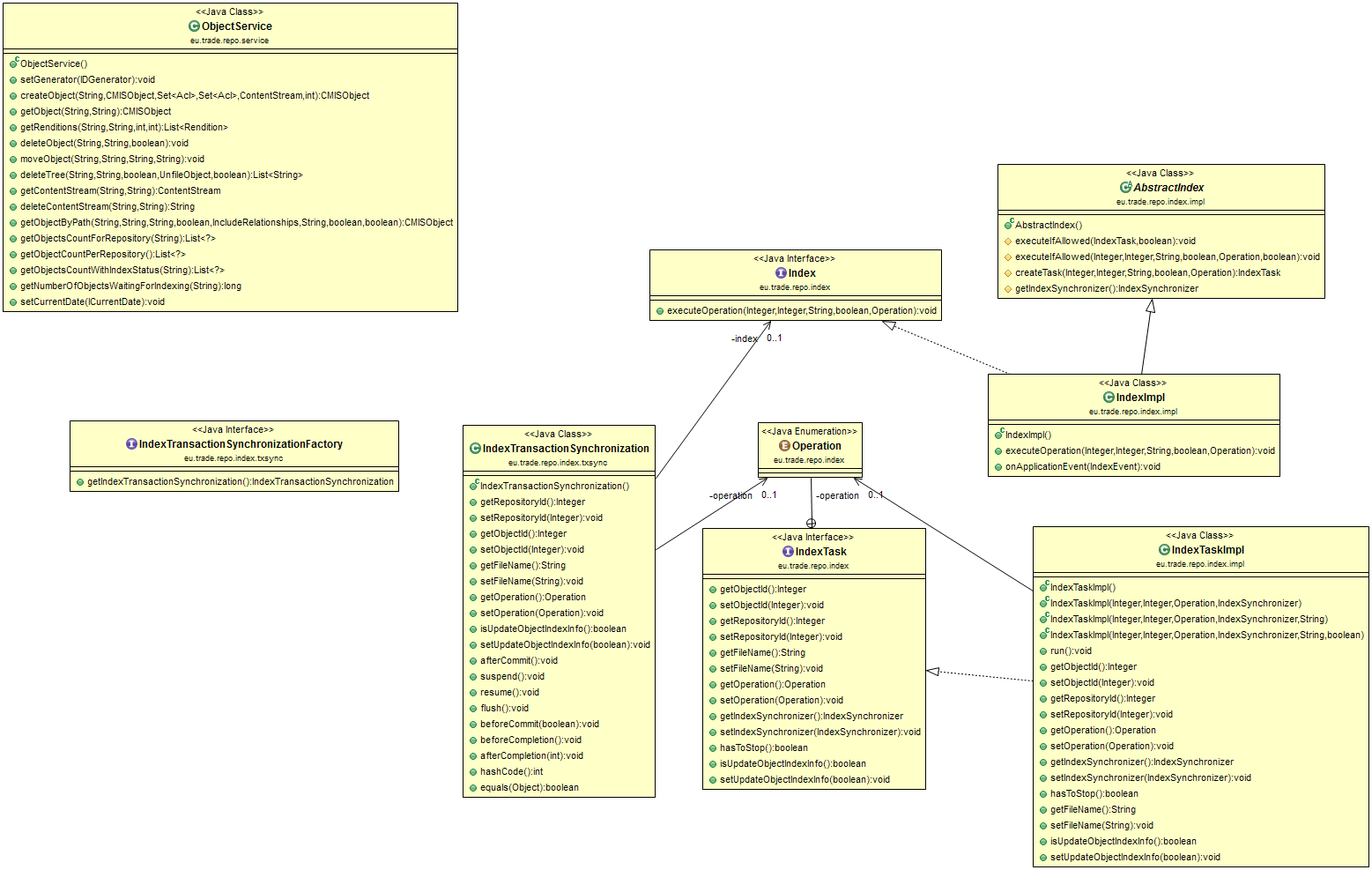


Figure 4. Background index

