COMP3021 2023 Spring

PA3

Maryam MASOUDIAN(mamt@cse.ust.hk)
Yibo Jin(yjinbd@cse.ust.hk)

Objectives

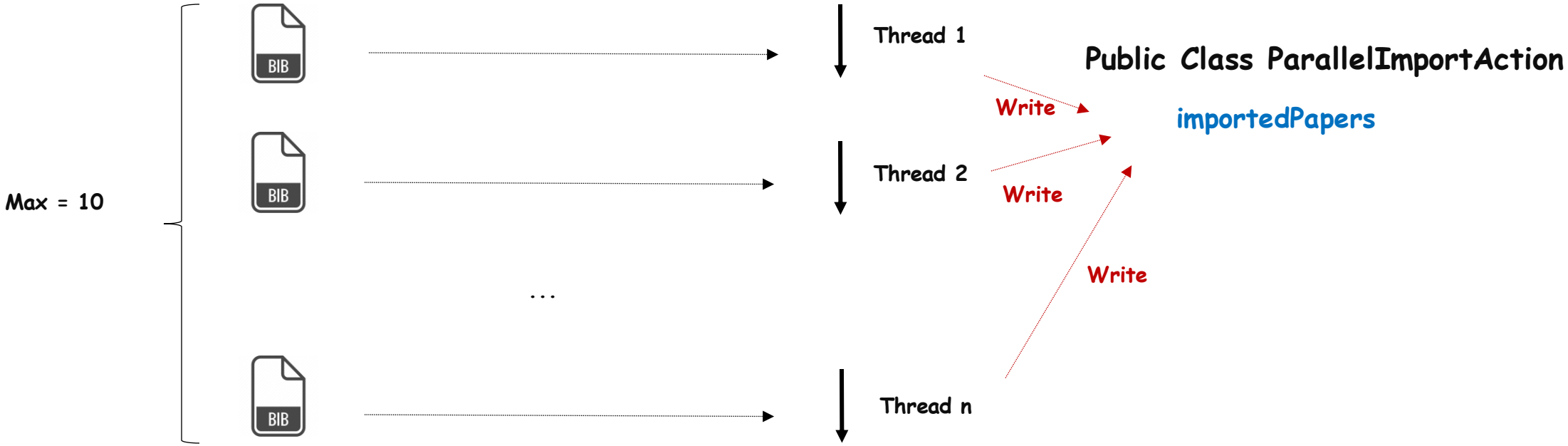
- In this part of the project, you are asked to practice multithreading and synchronization in Java programming.
- You must implement the following requirements:
  - Parallel importing of bib files
  - Rapid Multiple keywords search
  - Concurrent Modifications on Labels
  - Efficient Query Processing

Requirement 1: Parallel Importing of Bib Files

You should add a new functionality to import multiple bib files in parallel

1. Implement the functional interfaces of the class `ParallelImportAction` to add a new kind of action to the MiniMendeley to import multiple bib files in parallel.
2. Create a suitable data structure named `importedPapers` in the `ParallelImportAction` class to store the list of papers retrieved from the files.
3. Utilize the method `userInterfaceForParallelImport` in the MiniMendelyEngine class to receive the absolute path of the files from the user.
4. Create an instance of `ParallelImportAction` to simultaneously import the papers from files.
5. Implement the function `processParallelImport` to create multiple threads to read from files, import the bib files in parallel and store the imported results in `importedPapers` .
6. Add all the files stored in `importedPapers` to `paperBase` data structure used for storing the papers.

Requirement 1: Parallel Importing of Bib Files



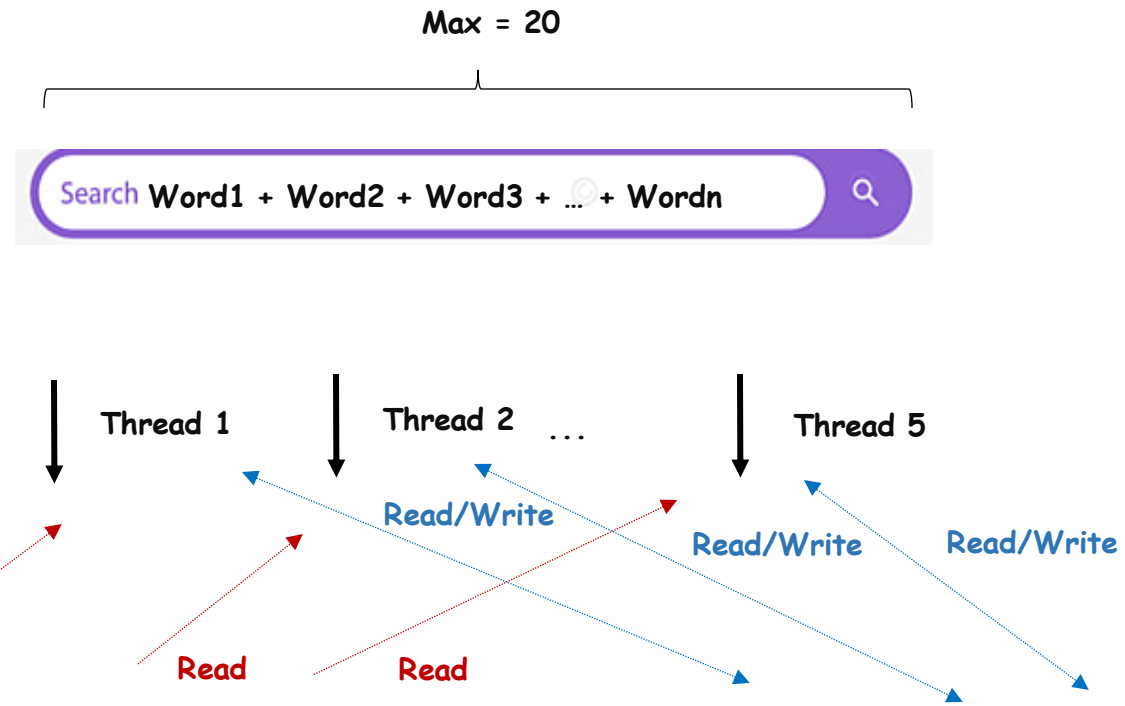
Note: The data structure `importedPapers` is shared between the threads. You must ensure that all the threads have equal access to it.
Note: A user can only import 10 files at a time.
Note: You can only use lock, wait, notify and notifyAll to synchronize the threads.

Requirement 2: Rapid Multiple Keywords Search

You should add a new functionality to search multiple keywords rapidly

1. Implement the functional interfaces of the class `SearchMultipleKeywordsAction` to add a new kind of action to the MiniMendeley to perform a rapid search for multiple keywords in *title*, *abstract* or *keywords* of papers.
2. Create an ArrayList named `results` in `SearchMultipleKeywordsAction` class to store the results of the search.
3. Utilize the method `userInterfaceForMultipleKeywordsSearch` in the MiniMendelyEngine class to receive the words from the user.
4. Create an instance of `SearchMultipleKeywordsAction` for simultaneously searching the papers for the words.
5. Implement the function `processMultiKeywordSearch` to create multiple threads to perform the search and save the results in `results` .

Requirement 2: Rapid Multiple Keywords Search



Pay Attention!

Threads < #Words

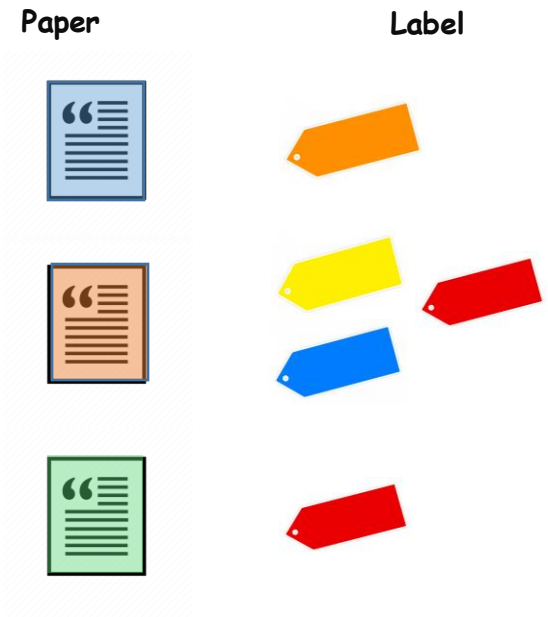
Note: *abstract, title* and *keywords* of a paper should be searched for the entered words.
Note: You can only use 5 threads for the searching process.
Note: Try to use all the threads effectively to perform the search operations with high performance.
Note: You should store the results in `results` that is shared between the threads.
Note: You can only use semaphore, lock, wait, notify and notifyAll to synchronize the threads.

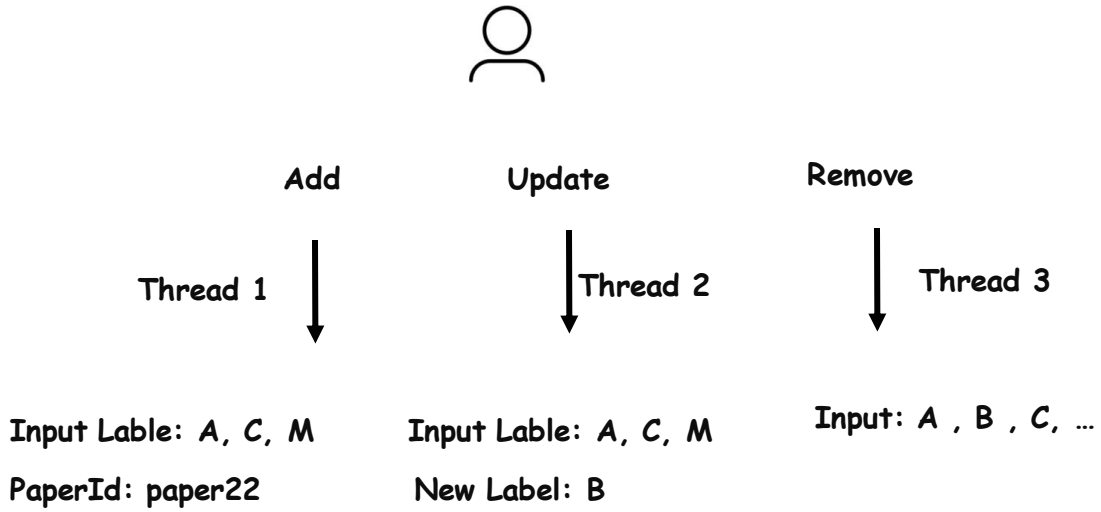
Requirement 3: Concurrement Modifications on Labels

You should add a new functionality to concurrently add, update or delete labels

1. Implement the functional interfaces of the class `LabelActions` to add a new kind of action to the MiniMendeley to perform add, update or remove operations on labels.
2. Utilize the method `userInterfaceModifyLabels` in the MiniMendelyEngine class to receive the type of the action with inputs from the user.
3. Use a queue named `labelActionsQueue` in MiniMendelyEngine class to store the actions entered by the user and add the entered action to it.
4. Implement three separate threads for handling each of the add, update or delete operations on the labels.
5. Implement the function `processLabelOperation` to poll an action from the queue and pass it to the thread corresponding to the operation type (add, update or delete).

Requirement 3: Concurrent Modifications on Labels





HashMap<String, paper> paperBase

Note: You must create three threads for performing any of the mentioned actions on labels that are executed in the background of the program. So, the user can interact with the program and perform more actions interactively.
Note: Once an action is completed, the user must be notified.
Note: Make sure you take advantage of the benefits of multithreading in your implementation for performing all kinds of operations in parallel.

Requirement 4: Efficient Query Processing

You should add a new functionality to concurrently read from a large file and process each query at each line of it

1. Implement the functional interfaces of the class `QueryAction` to add a new kind of action to the MiniMendeley to read and queries read from a large file consisting of thousands of queries
2. Complete the class `Query` and use it to represent a query that can be in any form of
   - ADD,object, newValue
   - UPDATE, object, condition, newValue
   - REMOEVE, object, condition

   An `object` refers to the object target of the query
   1. Paper
   2. An element of a paper

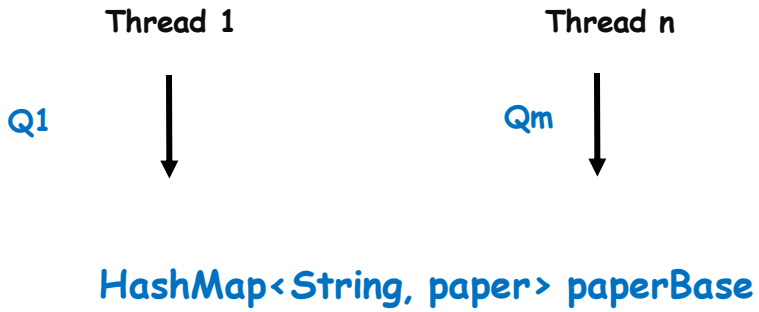
   A `condition` is an equality comparison

   A `newValue` includes the new value of the object and must comply with its type
3. Utilize the method `userInterfaceConcurrentQueryProcess` in the MiniMendelyEngine class to receive the absolute path of the file.
4. Create an instance of `processConcurrentQuery` and use threads to read from the file and store the queries in an ArrayList and process them concurrently in separate threads.
5. Make sure the data structures `paperBase` , that contains the paper lists, is consistent between threads

Requirement 4: Efficient Query Processing

Example of Queries:

Q1 Update, author, John H, Tom H
Q2 Delete, paper, paper1
Q3 Update, pages=112-113, 114-115
Q4 Update, title, paperId=paper3, A New Study on Threads
Q5 Add, paper3, { author="Tom", Title=" ...", … }



Note: You can use as many threads as you want for this task.
Note: You must ensure that your implementation satisfies the highest efficiency in processing all the queries within a file with thousands of queries.
Note: You can only use semaphore, lock, wait, notify and notifyAll to synchronize the threads.