Description

The existing application should be extended to support asynchronous processing of JSON files from a file-system folder.
Scheduled job is triggered when the application starts. Once per `SCAN_DELAY` seconds, scheduler runs `THREAD_COUNT` threads to grab JSON files from some specific folder and its sub-folders recursively.
If a file contains a list of all valid objects (schema from REST API Advanced task), all of them should be saved to a database. Then JSON files are deleted. In case a validation, input-output, JSON processing or any other error occurs – invalid files should be moved to an error folder that is located inside a specific folder with respect to file names collisions.
For demonstrating this task, a separate utility should be implemented. When it starts, it populates a specific folder with files in accordance with the following algorithm.
For `TEST_TIME` seconds:

1. Creates `SUBFOLDERS_COUNT` sub-folders (average depth = `SUBFOLDERS_COUNT` / 3).
2. Populates each folder, including the root one (`SUBFOLDERS_COUNT + 1`), with JSON files once per `PERIOD_TIME` seconds with `FILES_COUNT` files using one thread per folder (no thread pool needed here).
3. `FILES_COUNT` consists of the following data with the following distribution:
    a. count of not valid files:
        i. wrong JSON format : 1x,
        ii. field names (in the middle of a list): 1x,
        iii. non-valid bean (in the middle of a list): 1x,
        iv. violates DB constraints (in the middle of a list): 1x,
    b. valid files: 16x.
4. Each file should have at least 3 ~~certificates~~ news and each ~~certificate~~ news should have a unique ~~name~~ title across the whole application.
5. When the utility finishes its work, it should check database and file system to make sure that all files were processed correctly. Expected increase of count of files in the error folder and expected increase of rows in a table in the database should be equal to the actual ones.

Technical requirements

1. Only Java classes should be used. It is prohibited to use Spring (@Async, @Scheduled) and Quartz framework features for thread management.
2. Folder path, folder listening timeout, count of threads and other parameters have to be configured in a property file.
3. For demo please use:
    a. `TEST_TIME = 10`
    b. `FILES_COUNT = 1000`
    c. `PERIOD_TIME = 0.08`
    d. `SUBFOLDERS_COUNT = 9`
    e. `SCAN_DELAY = 0.1`
4. Valid processed files should be removed.
5. Invalid documents or documents that were not processed due to internal error have to be placed to an error folder in the root of the shared folder.
6. JSON documents can be in subdirectories thus recursive loading have to be implemented.
7. Each document should be processed in a separate thread from a thread pool.
8. Thread pool has to have fixed capacity and configured through the property.

Materials

This section contains links to materials recommended for self-study:

1. https://en.wikipedia.org/wiki/Producer–consumer_problem
2. https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html
3. https://urvanov.ru/2016/05/17/java-8-файлы-nio-2/
4. https://habrahabr.ru/company/luxoft/blog/157273/