

Report on the coding Task No. 2 & Task No. 1 of BCE

The goal of this report is to provide the detailed personal feedbacks on the coding task 2 & task 1 while targeting the question *“What worked well and Where were the main difficulties?”*

Task No. 2

My initial idea to accomplish the Task No. 2 can be summarized in the following sentence:

First of all, I wanted to have MVP0 (Minimum Viable Program Version 0) that realizes a basic required functionality and then, if the time left, re-factorize the code in order to have better structured & clear code.

To aim this goal, I built a following step-by-step plan

- 1) Analyze the structure of txt-file
- 2) Connect to database, check it and prepare tables for following data insertion
- 3) Create a mini test txt-version of the initial text data file and make this file the shortest but the trickiest possible, in order to capture all possible cases in given big file.
- 4) Develop a basic one row insertion into database for both tables and check that a mini-insertion works well for my particularly short but complicated version of my mini text file.
- 5) Enlarge this version in order to have another two methods that can handle massive data insertions into the tables
- 6) Develop the database two criteria search method for DB.
- 7) Re-test 1-6 aka MVP0 & reorganize the code.
- 8) Add extra search methods

After analyzing the given txt file, I came to conclusion that my DB should have two tables' SERIES & EPISODE. I have connected to the DB and created tables and appropriate columns. The difficulties were following:

- 1) No one can guarantee that the table EPISODE always has a link to SERIES in a source file, the only linking criterion was `$(SERIES_NAME)`, but no one can guarantee neither the identity and a right spelling in every possible case of EPISODE -> SERIES relation.
- 2) No one can also guarantee that the order is consistent (“SERIES” goes always after “EPISODE” or another “EPISODE”)
- 3) I see the possible troubles with the case of suspended flag (for example the first glance on 10000 cases) for the EPISODE table case I didn't see anything else, but always `[]` in contrary to `"$(SERIES_NAME)" $(SUSPENDED_FLAG)`

Taking into account 1) – 3) I created *testfirsttwi.txt* file, I put as a test case for EPISODE table a phrase “suspended” for flag = false, for example: `???? "#Adulthood" [SUSPENDED] (????-)`

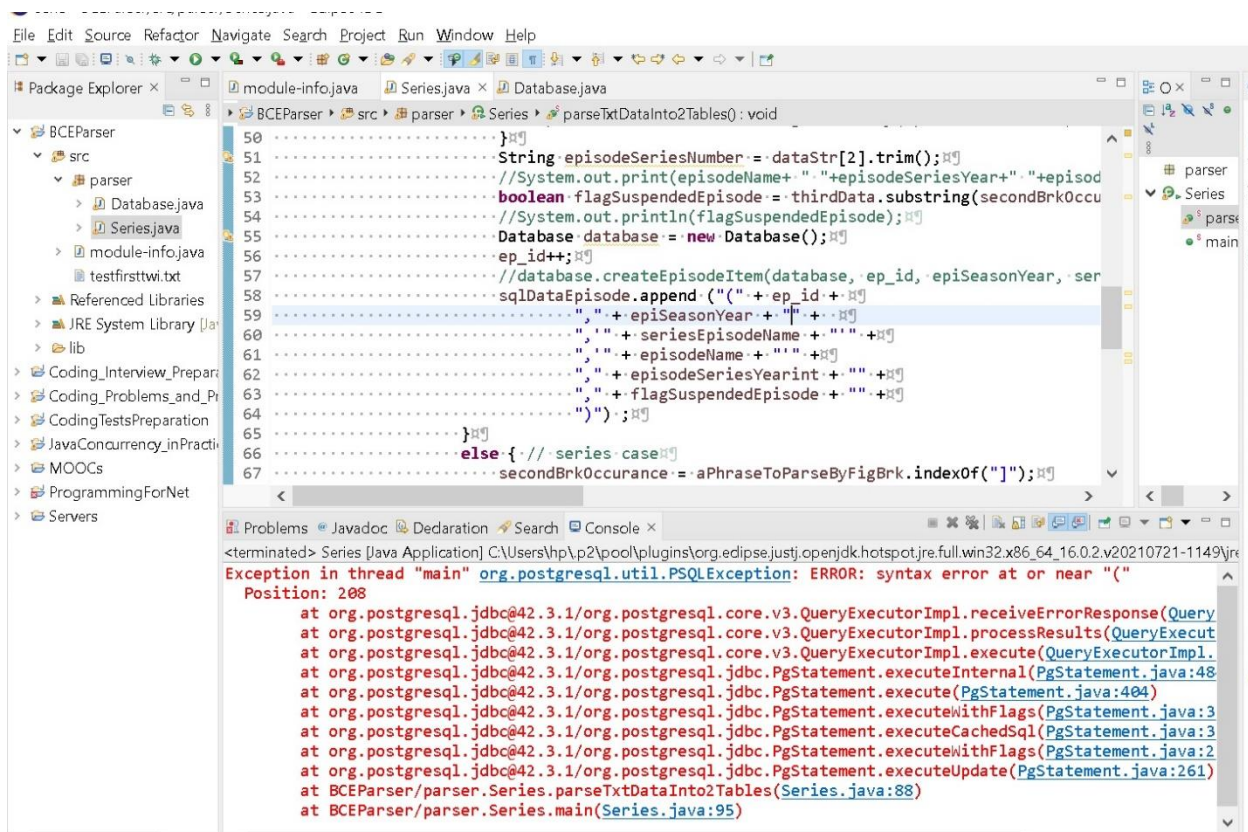
After quick viewing of first 10000 examples, I decided to do the following implementation:

My solution reads the txt file, skips the first descriptive part. Based on the structure of each line the code distinguishes EPISODE from SERIES data lines; therefore, a program should not care about the order of written data lines. The structure of a line itself is sufficient, so even if somebody mistakenly shuffled it, it should have no issue. Therefore, I keep two independent IDs for two tables and I form two appropriate independent one-line insertion requests for two tables.

After resolving issues with schema *public* and correcting the issue of \", I had my two well basic insertion into a table methods : `createSeriesItem()` and `createEpisodeItem()` that I run upon a given big txt file in a reading cycle.

No surprise for me that *max_level* DB connection was set equal to 100, so I have inserted only 100 first lines into DB.

So, I have dropped the first EPISODE table and started to work on the proper bulk massive insertion into DB. This struggle stopped me on:



The screenshot shows an IDE with a Java project. The Package Explorer on the left shows the project structure. The main editor displays the `Series.java` file, which contains a `parseTxtDataInto2Tables()` method. The code is using a `Database` class to insert data into a PostgreSQL database. The console window at the bottom shows a `SQLException` with the message: `ERROR: syntax error at or near "("`. The stack trace indicates the error occurred in the `executeUpdate` method of the `PgStatement` class, which was called by the `Series` class.

Task No. 1 scheduled on 26.10.2021

My initial plan to accomplish the Task No. 1 was similar to the plan for the Task No. 2

- 1) Bild a MVP0;
- 2) Add the basic testing

- 3) Enhance the code structure
- 4) Add supplementary methods to the lib.

I read the examples provided on an API Main WebSite and started to implement it. I have generated basic maven lib with non-regression test inside and started to adapted it to the requirements of Task No. 1. The most difficult was after getting an API reply, was to get through the right accessors' hierarchy, the right fields, like it was in provided example. After that, to put data in TVMazeContent object and finish the search method was easy. Having done that, I have quickly added the 3 basic nonreg-tests of *searchContent()* method and tested that all via console.

```
C:\dev\bce\apilib>mvn verify
[INFO] Scanning for projects...
[INFO] -----< org.bce.api.lib:apilib >-----
[INFO] Building apilib 0.0.1
[INFO] -----[ jar ]-----
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ apilib ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\dev\bce\apilib\src\main\resources
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ apilib ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ apilib ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\dev\bce\apilib\src\test\resources
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ apilib ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ apilib ---
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running org.bce.api.lib.TVMazeImplTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.575 s - in org.bce.api.lib.TVMazeImplTest
[INFO] Results:
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```