Java Developer coding challenge

Input data

1. CSV file.
   Columns definition:
   orderId, amount, currency, comment

   Entry example:
   1,100,USD,order payment
   2,123,EUR,order payment

   NOTE: all columns are mandatory

2. JSON file.
   Entry example:
   {"orderId":3,"amount":1.23,"currency":"USD","comment":"order payment"}
   {"orderId":4,"amount":1.24,"currency":"EUR","comment":"order payment"}

   NOTE: all fields are mandatory

Output data

{"id":1, "amount":100, "comment":"order payment", "filename":"orders.csv", "line":1, "result":"OK" }
{"id":2, "amount":123, "comment":"order payment", "filename":"orders.csv", "line":2, "result":"OK" }
{"id":3, "amount":1.23, "comment":"order payment", "filename":"orders.json", "line":1, "result":"OK" }
{"id":4, "amount":1.24, "comment":"order payment", "filename":"orders.json", "line":2, "result":"OK" }

- id - order identifier
- amount - order amount
- currency - order amount currency
- comment - order's comments
- filename - source file name
- line - line number from source file
- result - parsing result
  - OK - in case parsing have succeed,
  - or error description otherwise.

Task description

Write a Java program that parses incoming data from a file, convert it into output format and print result to stdout.

1. Solution should be as simple as possible. Taking into account the application could be maintained by other less experienced developers.
2. The application must be implemented using the **Spring framework**.
3. The source code of the application should be formatted as a **maven** project and posted on **GitHub**.
4. It is allowed to use dependencies only from **public** repositories.
5. Building the final application should be done with the command:
   mvn clean install
6. The application must be console-based.
   Example of a run command:
   java -jar orders_parser.jar orders1.csv orders2.json … ordersN.json
   - consider **orders1.csv**, **orders2.json** and **ordersN.json** are files to parse.
7. The result of the execution should be output to **stdout** stream.
   Note: only output data should go to stdout, no logs should be there.

8. Parsing and converting should be done in parallel in multiple threads.
9. It is necessary to provide correct error handling in the source files.
   For example, instead of a number, the file may have a string value in the **amount** field.
10. It is allowed to use language tools no higher than Java 8.
11. Consider the possibility of adding new input data formats. For example: XLSX