Table of Contents

Setup instructions

After the extaction of `.zip` folder, you will the following directory structure.

```
.
├──── build  ①
│     └──── data-pipeline-0.1-jar-with-dependencies.jar ②
├──── data  ③
│     ├──── hotels_large.csv
│     ├──── hotels_small.csv
│     ├──── hotels_small_noheaders.csv
│     └──── hotels_small_partialheaders.csv
├──── javadoc ④
│     ├──── allclasses-frame.html
│     ├──── allclasses-noframe.html
│     ├──── com
│     ├──── constant-values.html
│     ├──── deprecated-list.html
│     ├──── help-doc.html
│     ├──── index-all.html
│     ├──── index.html ⑤
│     ├──── overview-frame.html
│     ├──── overview-summary.html
│     ├──── overview-tree.html
│     ├──── package-list
│     ├──── script.js
│     ├──── serialized-form.html
│     └──── stylesheet.css
├──── pom.xml
├──── readme ⑥
│     ├──── README.html ⑦
│     ├──── README.pdf ⑧
│     └──── imgs
└──── src ⑨
      ├──── main ⑩
      └──── test ⑪
```

① build directory

② the library

③ sample data

④ source documentation folder

⑤ the javadoc main index file

⑥ documentation of the project

⑦ this readme file

⑧ the same readme file in PDF

⑨ main source directory

⑩ main source files

⑪ test cases folder

Dependencies

- Java, JDK version 1.8+. OpenJDK or Oracle either of which will work.
- Maven v3.0 or above
- No platform dependency.

Compilation

```
$ mvn clean package clean
```

the above command will download the required dependencies from the central repository and assembles the project with the required dependencies and bulds in `/target` folder. `dataconverter-0.1-jar-with-dependencies.jar` is the jar with the required dependencies.

Test Usage

for testing the library, I have added some sample data in `/data` folder. The test usage

```
$ java -jar build/data-pipeline-0.1-jar-with-dependencies.jar --read-format csv
--write-format [xml|json|md|sql|yaml] data/hotels_small_noheaders.csv --no-headers
...
$ java -jar build/data-pipeline-0.1-jar-with-dependencies.jar --read-format csv
--write-format [xml|json|md|sql|yaml] data/hotels_small.csv
```

Data pipeline

```
A Schematic overview of the Data-pipeline library:

                                                                                ...
                          [DataRecord:X]                                        [DataRecord]
[File] → [DataParser]   → [DataRecord]   → [DataProcessor] ... [DataProcessor] → [DataRecord] → [DataWriter] → [File]
             ↑            [DataRecord]                                        [DataRecord]
                                                                                ...
[FieldValidator][FieldValidator] ..


where
     [DataField] :     a datafield describes an entry into each individual record. it contains a fieldValue and fieldName associated with it.
     [FieldValidator] : a fieldValidator is used by the parser to validate each datafield on data record that it observes while parsing.
                        we can define what criteria to place to validate a field and a defaul fallback value to let it fallinto when the validation fails.

     [DataRecord] :    [DataField] [DataField] ... [DataField]
     [DataRecord:X] :  Header record describing the fields
```

Figure 1. an overview of the design of this library

Why so complicated pipeline?!

a question might arise why design such a complicated pipeline when the goal is to *convert the data from one format to another*. Here are some of my rationales or design goals:

- Should be easy extended
- Support for custom datatypes.
- Support for pre & post- processing of data before and after reading data.
- Support for multiple dataformats both for reading and writing data
- Support for data-sanatization before processing.

alright, how do I use it

- Get started?
  - Check `com.tckb.usage.TestUsage` for sample usage
- Have a new dataformat you want to use?
  - **extend** `com.tckb.data.parser.RecordParser` and `AbstractRecordWriter` to implement your own custom dataformat
  - as an example, I have a written a custom writer and parser in `com.tckb.usage` please check.
- How do I use it for my custom data type
  - your custom datatype **must** `implements SerializableData<?>` for the usage. Please write a resonable logic for unimplemented methods. as a sample, check `com.tckb.usage.Hotel` *

all great! but has it been tested?

I have tested with some basic `JUnit` test cases for **PoC** and seems to be working. But, hey I had just 2 days to design, code, test and document! cut me some slack ;)

Gotchas!

- the library has been designed keeping **KISS, YAGNI & DRY** principles with focus on extendability and stability in mind. So, please expect trivial, non-optimized code at places.
- the library is not ***THREAD SAFE*** ! but easily can be modified/extended to be one. please be aware of this while testing the code