Analysis Report: Java Code `CsvToJsonConverter`

Monday 31st March, 2025

Contents

1 Introduction

This report describes the Java code provided in the `converter` package (`CsvToJsonConverter`). Its objective is to explain in detail the purpose of the code, its functionality, and provide a higher-level overview of its implementation based on the latest provided version. The code is designed to read data from a CSV file, interpret it according to specific business logic, transform it into a structured data model comprising specific entities (`Item`, `Document`, and related types) and relationships, and finally serialize this model into a JSON file. It incorporates logic to handle missing or invalid data by assigning the string literal ‘"N/A"‘ to specific fields under defined conditions.

2 Purpose of the Code

The primary purpose of the `CsvToJsonConverter` code is to convert a data catalog stored in a tabular CSV (Comma Separated Values) format into a hierarchical and structured JSON (JavaScript Object Notation) format.

More specifically, the code aims to:

- Read data from a specific CSV file (`HCLEcatalog.csv`).
- Interpret each CSV row as representing a primary entity, classifying it definitively as either an **`Item`** or a **`Document`** based on specific criteria.
- Extract and map data from CSV columns to the fields of well-defined Java objects (POJOs - Plain Old Java Objects), representing Items, Documents, Persons, Organizations, Categories, and Materials.
- Apply data validation logic: If certain fields required for `Document` or `Item` identification or common fields are invalid or missing (null/empty/"None"), assign the literal string **`"N/A"`** to specific target fields (`toC`, `extent`, `serialNum`, `bibCit`, `created` for Documents; `partNum` for Items). Other fields are populated only if valid, otherwise left null.
- Identify and create related entities such as persons (`Person`), organizations (`Organization`), categories (`Category`), and materials (`Material`) from data in other columns (e.g., `Creator`, `SubjectTop`, `Material`).
- Establish meaningful relationships between these entities (e.g., an `Item` "belongsTo" a collection, a `Document` is "developed by" a `Person`, an `Item` is "madeOf" a `Material`). Note that relationship identifiers involving Items now incorporate the `partNum`, which might be "N/A".
- Ensure the uniqueness of the extracted entities and relationships (avoiding duplicates) using Java Sets.
- Generate a log file (`parsing.log`) that tracks the processing status and reports any errors or warnings for each CSV row.
- Produce a final JSON file (`data.json`) representing all unique entities (Items, Documents, Persons, etc.) and their relationships in a well-defined structure, suitable for further processing, analysis, or integration with other systems.

In summary, the code acts as a bridge between a tabular CSV format and a structured JSON format, applying domain-specific classification logic (Item vs. Document) and specific rules for handling missing/invalid data ("N/A" assignment).

3 Main Functionality (What it Does)

The code executes a series of logical steps to achieve its purpose:

1. **CSV Reading and Parsing:**
   - Opens the specified CSV file (`HCLEcatalog.csv`) using UTF-8 encoding.
   - Utilizes the Apache Commons CSV library to interpret the file.
   - Recognizes the first row as the header, ignoring case in column names.
   - Trims leading and trailing whitespace from each read value.
   - Iterates over each record (row) in the CSV file, excluding the header.
2. **Row Validation and Data Extraction:**
   - For each row, extracts the value from the `IdNum` column. If it is missing or effectively empty, the row is skipped, and an error is logged.
   - Extracts values from all other columns defined in the header and stores them in a map (`Map<String, String>`) for easy access. Values are stored as extracted (null if missing).
3. **Determining the Main Entity Type (Document or Item):**
   - By checking *ToC*, *Extent*, *SerialNum*, *BibCit* and *PartNum* we are able to determine if the row represents a `Document` or an `Item`.
4. **Creating the Main Entity Object (Document or Item):**
   - Instantiates a Java object of the determined class (`Document` or `Item`).
   - Populates the object's fields with values extracted from the corresponding CSV columns, applying specific "N/A" logic:
     - **For Documents:**
       - Sets `toC`, `extent`, `serialNum`, `bibCit` fields to the string `"N/A"` if the corresponding CSV value is null, empty, or "None". Otherwise, uses the valid CSV value.
       - Sets `created` field: Uses the value from `Created`, falls back to `DateCR`, then sets to `"N/A"` if the resulting value is null, empty, or "None". Otherwise, uses the valid date string.
       - Sets `copyrighted` field: Converts 'y' to `true`, 'n' or '0' to `false`. If the value is missing, invalid, or "None", the field remains `null` (no "N/A" applied).
     - **For Items:**
       - Sets `partNum` field: Sets to the string `"N/A"` if the corresponding CSV value is null, empty, "None", or consists of only a single digit. Otherwise, uses the valid CSV value.
       - Sets `conditionNts` field only if the corresponding CSV value is valid (not null, empty, or "None").
     - **For Common Fields (Description, DescComment, WherMade):** Populates these fields on the `Document` or `Item` object only if the corresponding CSV value is valid (not null, empty, or "None").
5. **Identifying and Creating Related Entities:**

- **Material:** If the `Material` column contains 'papr', 'digi', or 'mix', creates a `Material` object ("paper", "digital", "mix").
- **Category:** If the `SubjectTop` column has a valid value, creates a `Category` object.
- **Person/Organization (from Creator, Contributor, AddlAuth):** Logic remains the same as previous description (using regex, `parsePerson`, `processContributorField`) to create `Person` or `Organization` objects.

6. **Uniqueness Management:**
   - Utilizes `HashSet` collections for `Item`, `Document`, `Person`, `Organization`, `Category`, `Material`, and `Relationship`.
   - Ensures uniqueness based on the `equals()` and `hashCode()` methods in the POJO classes.
7. **Relationship Creation:**
   - `belongsTo` (Collection -> Entity): Created for every valid `Item` or `Document`. The object identifier includes the title and, for Items, the `partNum` .
   - `madeOf` (Item -> Material): Created only for `Item`s with an identified `Material`.
   - `describe` (Category -> Document): Created **only** if the main entity is a `Document` and a `Category` was identified.
   - **Creator Relationships** (`developed` / `produced`): Created **only** if the main entity is a `Document` and a `Creator` (Person/Organization) was identified.
   - **Contributor/AddlAuth Relationships** (`developed/produced/collaborated`): Created for both `Item`s and `Document`s if contributors/authors are identified.
8. **Logging:**
   - Continues to log processing status, errors (missing IdNum/Title), and warnings (unrecognized Material code, failed person parsing) to `parsing.log`.
   - The summary log at the end
9. **JSON Output Generation:**
   - Collects unique `Item`s and `Document`s (and other entity types) into the `Entities` container.
   - Filters relationships based on the validity of essential fields (as before).
   - Creates the root `JsonOutput` object.
   - Serializes the `JsonOutput` object to `data.json` using Jackson, producing formatted JSON. Fields with '"N/A"' values are included as strings.
10. **Error Handling:**
    - Continues to use try-with-resources and catch blocks for robust file handling and error reporting.

4 Implementation Overview (How it Works)

This section provides a high-level view of how the code achieves the updated functionality.

- **External Libraries:** Still relies on Apache Commons CSV and Jackson Databind.
- **POJO-Based Structure:** The data model uses POJOs (`Item`, `Document`, `Person`, etc.). The `Artifact` class now primarily serves as a **base class** providing common fields (like `title`, `description`) inherited by `Item` and `Document`. Specific fields in `Item` and `Document` are now defined to potentially hold the string ‘"N/A"‘. `equals()`/`hashCode()` are essential for uniqueness in Sets.
- **Main Class and Control Flow:** The `CsvToJsonConverter` class orchestrates the process. The core loop iterates through CSV records. The entity type determination logic is simplified to a binary choice (Document or Item). Logic for populating fields now includes checks for assigning ‘"N/A"‘ based on the rules.
- **Conditional Logic and Regex:** Entity classification is now based on the presence of specific Document-related fields. If not a Document, it's an Item. Part number validation for Items also uses a simple regex check (`^\d$`). Regex for parsing person names remains the same.
- **Use of Standard Collections:** Continues to use `Map` for row data, `Set` for uniqueness, and `List` for the final JSON structure.
- **File and Resource Management:** Try-with-resources and UTF-8 encoding are maintained for proper resource handling and character support.

5 Definition of Entities and Relationships

The code defines the following main data structures (POJOs), with modifications noted:

- `BaseEntity`: Abstract base class (unchanged).
- `Artifact`: Base class for Item and Document, providing common fields.
- `Item`: Subclass of `Artifact`. Represents a physical object. `partNum` field **can hold the string "N/A"**. `entityType` is "Item".
- `Document`: Subclass of `Artifact`. Represents a document. `toC`, `extent`, `serialNum`, `bibCit`, `created` fields **can hold the string "N/A"**. `entityType` is "Document".
- `Person`: Represents a person (unchanged). `entityType` is "Agent:Person".
- `Organization`: Represents an organization (unchanged). `entityType` is "Agent:Organization".
- `Category`: Represents a category (unchanged). `entityType` is "ContentDescription:Category".
- `Material`: Represents material (unchanged). `entityType` is "Material".
- `Relationship`: Represents a connection. Identifiers for Item subjects/objects now incorporate the potentially "N/A" `partNum`.

- `CollectionInfo`: Represents collection info (unchanged).
- `Entities`: Container grouping lists of entities.
- `JsonOutput`: Root object for JSON output (unchanged structure).

Jackson annotations control JSON serialization, including the omission of null fields and the inclusion of fields containing ‘"N/A"‘.

6 Estimation of the errors

The primary objective of this analysis was to perform a qualitative assessment of the data quality within the provided JSON file, identifying common error patterns, inconsistencies.
The assessment was conducted primarily through manual inspection and pattern analysis of the ‘Entities‘ and ‘Relationships‘ sections within the JSON data.

6.1 Creators and contributors

Here a resume about errors between Person and Organization

Classified as ↓ / Actual →	Person	Organization
Person	**n** (Correct)	**13** (Incorrect)
Organization	**71** (Incorrect)	**m** (Correct)

Table 1: Confusion matrix for People/Organizations classification

This suggests a potential issue in the original data source.

6.2 Summary

- **Entity Misclassification (People/Orgs):** 84 instances error estimated.
- **Placeholder/Incomplete Data:** Affects dozens to hundreds of records.
- **Formatting Inconsistencies:** Pervasive across relevant fields and records.