COMPUTER SCIENCE DEPARTMENT

Computer Science - Curriculum Artificial Intelligence

Project Assignment

Foundamentals of Artificial Intelligence

GraphBrain

Student: *Fontana Emanuele*

Academic Year 2024/2025

Contents

2 Exercise 1

2.1 Overview

This document provides a detailed description of the updates performed. The modifications have been structured into two main sections: firstly, the upload of various entities to specific classes, and secondly, the improvements proposed for the the interface.

2.2 Data Upload Details

- **Metal Slug Series - Main Games**: Approximately 10 main titles from the Metal Slug series *RETROCOMPUTING → VIDEOGAME*
- **Flight Simulator Series**: Around 10 flight simulation games. *RETROCOMPUTING → VIDEOGAME*
- **Street Fighter Series**: Roughly 8 distinct titles. *RETROCOMPUTING → VIDEOGAME*
- **Dragon Ball Games**: About 15 games. *RETROCOMPUTING → VIDEOGAME*
- **Pro Evolution Soccer Series**: Nearly 35 games including both current titles and their historical predecessors. *RETROCOMPUTING → VIDEOGAME*
- **Console Games**: Approximately 5 devices. *RETROCOMPUTING → CONSOLE*
- **Technology Vendors**: Details for 5 companies. *RETROCOMPUTING → COMPANY*
- **Peripheral Devices**: Information for about 15 mouse and keyboard devices. *RETROCOMPUTING → Input Device (Mouse, Keyboard)*
- **EXPO Events**: A list of approximately 35 events. *RETROCOMPUTING → Event*
- **Software Relationships**: For each videogame, a *producedBy* relationship has been established linking the software to the company that developed it.
- **Console Relationships**: For each console, the producing company has been recorded along with associated relationships to already existing consoles.
- **Peripheral Relationships**: For each mouse and keyboard device, the producer has been identified.
- **Geographical Data**: Inclusion of Matera and surrounding cities (approximately 30 locations).
- **Internet Protocols**: Updates include renaming 8 existing protocols and adding around 70 new entries. *RETROCOMPUTING → InternetProtocol*
- **Crapiata**: A traditional dish from Matera, described as a soup made with legumes and vegetables, albeit missing some ingredients. *FOOD*

- **Culinary Relationships**: Established relevant relationships associated with the aforementioned dish.

2.3 Interface Improvements

Several adjustments have been made to enhance the user interface:

- Incorporation of an HTML Date Type field for the insertion of dates.
- Modification of the relationship creation process to allow starting from either the Subject or the Object.

3 Exercise 2, with Cirilli Davide

3.1 Overview

This report describes the modification proposed for several ontologies and the Java code provided in the `CsvToJsonConverter` package. Its objective is to explain in detail the purpose of the code, its functionality, and provide a higher-level overview of its implementation. The code is designed to read data from a CSV file, interpret it according to specific logic, transform it into a structured data model (entities and relationships), and finally serialize this model into a JSON file.

3.2 Ontology Modifications

Here we will provide a brief overview of the changes made the ontology. The modifications are divided by domains and, for each domain, they are divided into two subsections: the first one is about the entities and the second one is about the reletionships

3.2.1 RETROCOMPUTING

Entities

- **StorageMedium**: We suggest to add a new value for *StorageMedium* called *Solid-State*. This value will be used to represent all the solid state storage devices such as SSD, USB pen drive and so on.
- **FPGA**: We suggest to add a new sub-class of *Device* called *FPGA*. This class will be used to represent all the FPGA devices, such as Microchip IGLOO Series
- **Videogame**: Since a videogame can be classified into multiple categories, We suggest to add an attribute to videogame called *Category* that will be a list of categorie such as FPS, Sport, RPG, MOBa and so on. The previously existing sub-classes of *Videogame* have been removed.
- **Preservation Project**: We suggest to add a new class called *PreservationProject* sub-class of *Artifact*. This class will be used to represent all the preservation projects that are related to retrocomputing for example *Internet Archive* or *MAME*. The new attributes are goal (mandatory) and description

- **Fix**: We suggest to introduce 2 new attributes to *Fix* which are *repairDifficulty* that can assume only 3 values (Beginner, Intermediate, Expert) and *documentationLink* that is a link to the documentation of the fix.

Relationships

- **supports**: We suggest to add this new relationship between *Device* (subject) and *Software* (object). This relationship will be used to represent the software that is supported by a specific device. The attribute is compatibilityNotes
- **compatibleWith**: We suggest to add Software (subject) and Component (object). This relationship will be used to represent the software that is compatible with a specific component.
- **supports**: We suggest to add this new relationship between *Device / OperatingSystem* (subject)and *Software* (object). This relationship will be used to represent the software that is supported by a specific device or operating system. The attribute is compatibilityNotes

3.2.2 FOOD

Entities

- **Beverage**: We suggest to add a new attribute called *Type* to indicate the type of beverage (alcoholic, non-alcoholic, etc.).
- **Menu Item**: We suggest to add a new attribute called *dietaryInfo* to indicate the dietary information of the menu item (vegan, vegetarian, gluten-free, etc.).
- **SensorialFeature**: Sensorial feature has been removed [1]
- **Restaurant**: We suggest to add the attribute *type* to indicate the type of restaurant (fast food, fine dining, etc.).
- **DietaryRestriction**: We suggest to add this new entity to represent the dietary restrictions that can be associated with a food item or menu item. The new attributes are name (mandatory) that can assume fixed values (vegan, vegetarian, gluten-free, etc.)
- **KitchenTool**: We suggest to add this new entity to represent the kitchen tools that can be used in the preparation of food. The new attributes are name (mandatory)

Relationships

- **contains**: We suggest to add this new relationship between *FoodBeverage* (subject) and *Nutrient* (object). This relationship will be used to represent the nutrients that are contained in a specific food or beverage. The attribute is quantity (mandatory) that can assume fixed values (low, medium, high).

[1]Sensorial Feature may be described as attributes in a relationships without a specific class.

- **requires**: The subject has been modified from *Artifact* to *KitchenTool*
- **describes**: New attributes have been added to express SensorialFeature

3.2.3 OpensScience

We've added the instruction <import schema "retrocomputing"> to the ontology to import the retrocomputing schema

Entities

- **Dataset**: We suggest to add new attributes: creationDate,license,format
- **Environment**: We suggest to add new attributes: type (whose values are Lab, Field or Virtual) and description
- **Author**: We suggest to add *Author* as a sub-class of *Person*

3.2.4 General

Entities

- **Material**: We suggest to add a new Category called *Material* to represent the materials that can be used to describe Item.
- **Document**: We suggest to add a new attribute called *ToC* to represent the table of contents of the document.
- **Item**: We suggest to add a new attribute called *conditionNotes* to represent the condition of the item

Relationships

- **madeOf**: We suggest to add this new relationship between *Item* (subject) and *Material* (object)

3.3 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 an entity (such as a physical object, document, person, organization, etc.) or containing information that defines entities and relationships.
- Extract and map data from CSV columns to the fields of well-defined Java objects

- Classify the main entities into specific types (e.g., `Item`, `Document`, `Artifact`) based on the presence or format of certain fields in the CSV.
- 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`).
- Ensure the uniqueness of the extracted entities and relationships (avoiding duplicates).
- 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 and their relationships in a well-defined structure, suitable for further processing, analysis, or integration with other systems (e.g., graph databases, web applications).

In summary, the code acts as a bridge between a tabular and potentially flat data format (CSV) and a structured, relational data format (JSON), applying domain-specific interpretation and classification logic based on the data within the CSV.

3.4 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 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.
   - Performs basic cleaning (e.g., removing residual quotation marks).
3. **Determining the Main Entity Type:**
   - Checks if the row represents a `Document` by verifying the presence of non-null/empty/"None" values in the `ToC`, `Extent`, `SerialNum`, or `BibCit` columns.

- If it is not a Document, checks if it is an Item by verifying that the PartNum column contains a non-null/empty/"None" value and that it is not a single digit.
- If it meets neither the criteria for Document nor Item, it is classified as a generic Artifact.
- Verifies the presence of the mandatory Title field. If missing, the row is discarded, and an error is logged.

4. **Creating the Main Entity Object:**
   - Instantiates a Java object of the appropriate class (Document, Item, or Artifact).
   - Populates the object's fields with values extracted from the corresponding CSV columns (e.g., title, description, partNum, toC, created, etc.).
   - Applies specific logic for certain fields:
     - For Document.created, uses the value from Created if present, otherwise falls back to DateCR.
     - For Document.copyrighted, converts 'y' to true, 'n' or '0' to false, leaving it null otherwise.
     - Excludes fields if their value in the CSV is null, empty, or "None" (handled by @JsonInclude(Include.NON_NULL) and isNullOrNone checks).
5. **Identifying and Creating Related Entities:**
   - **Material:** If the Material column contains 'papr', 'digi', or 'mix', creates a Material object with the corresponding name ("paper", "digital", "mix"). Unrecognized codes are logged as warnings.
   - **Category:** If the SubjectTop column has a value, creates a Category object using that value as the name.
   - **Person/Organization (from Creator):**
     - Attempts to interpret the value of the Creator column as a person's name using a specific regex (PERSON_REGEX_STRICT) and the parsePerson method.
     - If interpretation succeeds, creates a Person object with separate name and surname.
     - Otherwise (if the regex does not match or parsing fails), creates an Organization object using the original value as the name.
   - **Person/Organization (from Contributor/AddlAuth):**
     - Uses a dedicated method (processContributorField) to handle the Contributor and AddlAuth columns.
     - These fields can contain multiple names separated by commas, semicolons, "and", or "&".
     - The method identifies if the field contains a list of person names (PERSON_REGEX_LIST).
     - If so, it splits the string and attempts to parse each part as a Person. If parsing fails for a part, it treats that part as an Organization.

    - If the field does not resemble a list of persons, it treats the entire content as a single `Organization`.
    - Creates the corresponding `Person` or `Organization` objects.

6. **Uniqueness Management:**
    - Utilizes `HashSet` collections to store all entities (`Item`, `Document`, `Artifact`, `Person`, `Organization`, `Category`, `Material`) and relationships (`Relationship`).
    - Due to the correct implementation of `equals()` and `hashCode()` methods in the classes, adding an element to a `Set` only succeeds if an equal element does not already exist, thus ensuring uniqueness.

7. **Relationship Creation:**
    - For every valid main entity extracted from a CSV row, creates a **`belongsTo`** relationship:
        - Subject: The fixed collection "HCLE" (Type: `Collection`).
        - Object: The current entity (identified by title, or title + PartNum for Items). Type: `Item`, `Document`, or `Artifact`.
        - Includes the original `IdNum` value in this relationship's `number` field.
    - If a `Material` is identified for an `Item`, creates a **`madeOf`** relationship:
        - Subject: The current Item. Type: `Item`.
        - Object: The identified Material. Type: `Material`.
    - If a `Category` is identified for a `Document` or `Artifact`, creates a **`describe`** relationship:
        - Subject: The identified Category. Type: `Category`.
        - Object: The current Document/Artifact. Type: `Document` or `Artifact`.
    - If a `Creator` is identified for a `Document` or `Artifact`:
        - If it is a `Person`, creates a **`developed`** relationship. Subject: Person, Object: Document/Artifact.
        - If it is an `Organization`, creates a **`produced`** relationship. Subject: Organization, Object: Document/Artifact.
    - For each `Person` or `Organization` identified from the `Contributor` or `AddlAuth` fields, creates appropriate relationships:
        - From `Contributor`: Base type **`developed`** (becomes **`produced`** for Organizations). Subject: Person/Organization, Object: Current entity.
        - From `AddlAuth`: Base type **`collaborated`**. Subject: Person/Organization, Object: Current entity.

8. **Logging:**
    - Opens and writes to a log file (`parsing.log`) using UTF-8 encoding.
    - Logs the start of processing for each row, indicating the identified entity type, ID, and title.

- Logs critical errors (e.g., missing `IdNum`, missing `Title`) that cause the row to be skipped.
- Logs warnings (e.g., unrecognized `Material` code, failure to parse a name as a `Person`).
- Logs the completion of processing for each row.
- At the end, logs summary statistics on the number of unique entities and relationships found.

9. **JSON Output Generation:**
   - Collects all unique entities from the `Set`s and organizes them into lists within an `Entities` object. This object also contains the collection information ("HCLE").
   - Collects all unique relationships from the relationship `Set`.
   - Filters the relationships, discarding those that lack valid (non-null and non-empty) values for essential fields (`ontology`, `type`, `subject`, `subjectType`, `object`, `objectType`). The `number` field can be null.
   - Creates a root `JsonOutput` object containing the `Entities` object and the filtered list of `Relationships`.
   - Uses the Jackson library (`ObjectMapper`) to serialize the `JsonOutput` object into a JSON string.
   - Configures Jackson to produce formatted ("pretty-print") JSON output for better readability.
   - Writes the resulting JSON string to the specified output file (e.g., `data.json`) using UTF-8 encoding.

10. **Error Handling:**
    - Uses try-with-resources blocks to ensure automatic closing of files (CSV, log, JSON).
    - Catches `IOException` that may occur during file I/O operations or CSV parsing.
    - Catches generic `Exception`s to handle unexpected errors during processing.
    - Prints error messages to the standard error console (`System.err`) in case of severe exceptions.

3.5 Implementation Overview (How it Works)

This subsection provides a high-level view of how the code achieves the described functionality.

- **External Libraries:** The code relies on two main external libraries:
  - **Apache Commons CSV:** Used for robust and configurable reading of CSV files, automatically handling headers, delimiters, quoting, and value trimming.

  - **Jackson Databind:** A powerful library for handling the JSON format in Java. It is used to serialize the Java objects into the desired JSON structure, managing the mapping between Java fields and JSON keys via annotations (`@JsonProperty`, `@JsonInclude`, `@JsonIgnore`) and `ObjectMapper` configuration.

- **Structure:** The core of the data representation consists of Java Objects modeling the different entities (`Artifact`, `Item`, `Document`, `Person`, etc.) and relationships (`Relationship`). These classes contain public fields (or getters/setters, though omitted for brevity here) annotated with `@JsonProperty` to control JSON serialization. Inheritance (`Item` and `Document` extend `Artifact`, which extends `BaseEntity`) facilitates sharing common properties. The implementation of `equals()` and `hashCode()` is crucial for using `Set`s to ensure uniqueness.

- **Main Class and Control Flow:** The `CsvToJsonConverter` class contains the `main` method, which orchestrates the entire process. The main flow involves a loop iterating over CSV records. Inside the loop, helper methods are invoked to perform specific tasks like safe value extraction (`getRecordValue`), validation (`isNullOrNone`, `isNullOrEmpty`), name parsing (`parsePerson`), and handling fields with multiple contributors (`processContributorField`).

- **Conditional Logic and Regex:** Entity classification (Item vs. Document vs. Artifact) and the identification of Persons vs. Organizations rely on conditional logic (`if-else`) checking the presence and format of data in specific CSV columns. Regular expressions (Regex) are used purposefully (`PERSON_REGEX_STRICT`, `PERSON_REGEX_LIST`, `NAME_SPLIT_DELIMITER`) to identify specific patterns in person names and to split strings containing multiple names.

- **Use of Standard Collections:** The code makes extensive use of standard Java collections:

  - `Map<String, String>`: To store data from a CSV row accessible by header name.
  - `Set<E>`: Fundamental for collecting unique entities and relationships.
  - `List<E>`: Used in the final structure (`Entities`, `JsonOutput`) to hold the collections of entities and relationships for JSON serialization (Jackson works more naturally with lists for JSON arrays). Conversion from Set to List occurs before serialization.

- **File and Resource Management:** The use of try-with-resources ensures that files and streams (readers, parsers, writers) are closed correctly even if errors occur, preventing resource leaks. Specifying UTF-8 encoding is important for handling special characters correctly.

3.6 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.

3.6.1 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.

3.6.2 Artifacts, Documents and Items

A significant lack of clarity and consistency exists in the application of the ‘Document‘ and ‘Artifact‘ entity types.
Some of the entities classified as `Artifacts` possess characteristics typically associated with documents.

Classified as ↓ / Actual →	Document	Artifact
Document	a (Correct	**5** (Incorrect)
Artifact	**50** (Incorrect)	b (Correct)

Table 2: Confusion matrix for Artifact/Document classification

This suggests a potential issue in the original data source

3.6.3 Documents Representing Non-Documentary Items

Conversely, several items classified as `Documents` clearly represent physical hardware or software media. Examples include:

- `"Apple II Power Supply"`
- `"Apple II Plus Computer Assembly"`
- `"Apple Monitor III"`
- `"Hitachi Color Display"`

These are clear misclassifications based on the absence of values in the CSV file

3.6.4 Summary

- **Entity Misclassification (People/Orgs):** 84 instances error estimated.
- **Entity Misclassification (Doc/Artifact):** 55 instances error estimated..
- **Placeholder/Incomplete Data:** Affects dozens to hundreds of records.
- **Formatting Inconsistencies:** Pervasive across relevant fields and records.

Overall, while the People/Organization categorization errors are numerous, the inconsistent application of ‘Document‘ vs. ‘Artifact‘ and the widespread formatting/completeness issues represent major structural data quality challenges.