UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

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 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).

3.4 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 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`, `processContributorFi` 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.

3.5 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.

3.6 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"‘.

3.7 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.

3.7.1 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.

3.7.2 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.