Introduction

The **CompanyNews** project allows you to search in several news articles if a company, from a certain dataset, is mentioned or not.

The project uses Spring Boot to allow a clean structure about injection of control and testing.

Design choices

There were several ideas at the beginning of the project, and to assure the best result a good approach has to:

- Identify the companies with or without legal amendements (spa, srl etc.);
- Avoid too many false positives;
- Be efficient in time.

A common solution to all the different possibilities is to parse the articles taking a substring of all consecutive words with the first capital letter, because I don't know if it can be a person name, a place or a company. Then this substring will be compared with the strings in the companies dataset to find a possible match.

So some solutions were discarded:

- First, I tried to use the **Levenstein Distance** all over the dataset, without parsing or filtering it. But a lot of false positives were found.
  - This solution was also a lot time consuming. I tried to reduce time to divide the companies in 26 different arrays (one for each letter) but it hasn't helped with the false positives problem.
- Then I tried to create a Map of companies from the dataset lines. I've also implemented a **company extractor** to extract the info in the brackets (because a lot of times inside the brackets there is another way to call the same company) and assign the name to the same ID.
  - But this solution avoided a lot of possible companies due to the exact match of a company with the key of the map. A lot of companies can be called without the legal amendments for example.

Therefore I decided to implement the second solution with a **custom filter** to filter all the possible legal amendments and special word (e.g. "formerly known as") to reach a good compromise between avoiding false positive, matching the companies and do it in a good time (less than 10 seconds).

- Unfortunately I tried to create a dynamic filter to catch the legal amendments, but I didn't find a common pattern.
- It's far to be a perfect solution, but due to the dataset and the articles that have several uncleaned words, characters etc. it would be very difficult to reach a very high level of accuracy (for example sometimes inside the brackets there are company names or places, some companies without legal amendments are common names, etc).

Design

The project is designed in different packages:

- **Controller**: the main controller where is defined a REST API to start the search;
- **Filter**: the filter to remove the special words and legal amendments;
- **Finder**: the main Finder class, it constructs the Map of companies from the dataset and iterate all over the articles to find the mentioned companies;
- **Model**: a model class just to take the two file paths for the company dataset and articles directory;
- **Parser:** the two parser classes to get the company names both from the dataset and articles.

Filter

The filter deals with clean a string from all the legal amendments and special characters. It's useful for the company matching between two words. The methods are:

- **filter(String toClean):** it removes all the words contained in a Set of String from the toClean string;
- **filterComplexWords(String line):** it removes from the String line the word with two or more tokens.

DatasetParser

This class is used to parse all the companies from the csv dataset and return a Map where the key is the name of the company, and the value the ID.

- **getCompaniesFromDataset(String filePath):** the main method of the class to return the map.
  - First with a Scanner it gets all the file lines;
  - Then for each line, it detects all the company names (because in the same line there can be 2 or more ways to call a company)
  - Finally the names are filtered with the Filter class and added to the map.
- **getCompaniesInALine(String line):** this method gets a list of possible names for the same company:
  - until there are brackets, there are other possible names to refer to the same company
  - the method extractStrings() is called to extract the company name from a string.
- **extractStrings(String line, List<String> companiesInALine):** it extracts all the companies in a given String line and add them to a List of String companiesInALine.
  - It detects the separators ',' ';' 'or' to extract the possible names.

ArticleParser

The article parser class is used to parse the article and get all the possible company names.

The possible company names are all those words that start with the capital letter consecutively:

- **getPotentialCompanies(String article):** returns a Set of String containing all the possible company names.
  - The method uses a Tokenizer to tokenize the article;
  - The words are cleaned from special character with a regex;

    - A word is appended to a possible company name when it contains a special begin character (like ‘,’, ‘;’ etc.) and it begins with a capital letter or when I’ve already encountered a capital letter word and I have to continue until I reach a word with the first non-capital letter.

Finder

The main class to find all the companies mentioned in the articles.

- **getCompaniesIds(String pathOfArticles, String pathOfDataset):** it returns a list of companies IDs that are found in the articles:
    - It gets a list of Path for all the given articles, and the Map of companies from the dataset (key = company name, value = company ID) using the DatasetParser;
    - For each article, it starts a thread to find a company in a given article have a better time performance;
    - Finally, the list of company IDs are sorted to better readability.
- **getArticlesPaths(String pathOfArticles):** it returns a list containing all the paths of the articles from the “pathOfArticles” directory.
- **executeCompanyThreads(List<Path> articlesPaths, Set<Integer> companies, Map<String, Integer> companiesDataset):** it executes a thread for each article to search all the possible companies inside the article in the Map of companies:
    - Firstly, it gets all the possible company names from the article using the Article Parser;
    - Then it filters the word and if there is a matching with the Map, it adds the company ID to a Set of company IDs (to avoid duplicates).

Tests

There are Unit Test to test to checks if all works correctly. The code coverage is over 90%.