Home Assignment – Text Classifier

General Instructions - Please read before you start

- The assignment consists of several checkpoints which rely on each other. Therefore, **it is recommended to read the entire assignment prior to starting**.
- Using a high-level programming language (Java/C#/Python) is recommended.
- Your solution should be submitted via GitHub to a repository given to you. **If you do not have a repository name, please contact us and we will provide you with one**.
- You are encouraged to **use Git best practices** throughout your development, e.g., have each one of the checkpoints committed to their own branch.
- Following to submitting the assignment, you will review it together with the interviewer. During the interview **you might be asked to introduce new features to your solution**.

Objective – Creating a text classifier

Your assignment is to design and implement a command line tool that performs classification of textual content stored in files and folders into business domains.

Input

Your program will receive the following command-line arguments:

1. Path in the filesystem representing a file or a folder that should be scanned.
2. Path to a JSON file, representing the list of classification rules:

```
{
  "classification_rules": [
    {"domain": "financial", "indicators": ["credit card", "bank account"]},
    {"domain": "healthcare", "indicators": ["medical insurance"]}
  ]
}
```

As we can see, the file consists of a single object with a list field named `"classification_rules"`. Each element in the list represents a rule with the following fields:

`"domain"`: a string representing a business domain, like financial, healthcare, etc.

`"indicators"`: a list of strings, indicating that the scanned content belongs to the `"domain"`.

Output

The program should extract text from files under the provided path and to classify it into one or more domains. The text is classified into some "domain" if it contains one of corresponding "indicators".
For instance, given the classification rules above and the input file:

```
My medical insurance is billed from the bank account number 84NK 4CcOun7.
```

The expected output would be:

```
financial
healthcare
```

NOTES ABOUT DESIGN AND IMPLEMENTATION

The solution you build should be:

- **Well-structured**, **organized**, and **documented** (code and usage). It should be **maintainable**, and **reliable**, addressing **unexpected behaviors and corner cases** (e.g., missing or invalid user input).
- **Extensible**, e.g., introducing support for additional file formats should be simple. Remember that you might be asked to add features in the interview itself.
- **Memory efficient** - It should support very large input files and deep file system hierarchies. Yet you can assume that the number and the size of the classification rules will be not significant.
- It is recommended that you make sure that **each checkpoint is "production-ready"** prior to moving to the next checkpoint. **It is better to not finish all the checkpoints rather to finish them all with lower quality**.
- In case that you extend your solution beyond the stated requirements, please document what features were added.

And finally, you are **encouraged to use advanced language features and standard 1st/3rd party libraries**. There is no reason to reinvent the wheel, even in an interview assignment. If you have a doubt regarding using a specific library, don't hesitate to ask us.

CHECKPOINT I

In this checkpoint we create a tool that classifies a text of a single file. The file can be either a CSV or raw text. If it's a CSV file, parse it as such, otherwise parse the raw content. For example, for the same indicators as the example above, the following raw and CSV files:

```
first-header, second-header, third-header
"credit card", medical, insurance
bank, account, "hello, world"
```

```
You need to pay for your insurance, medical or other,
with either cash, credit card, or a cheque,
```

will produce the same output:

```
financial
```

CHECKPOINT II

Now we want to add additional functionality to the program to support either a single file or a folder. The program will return some domain if at least one of the files within the folder contains the relevant indicators. You can use existing frameworks for iterating the folder structure. Note that folders might also contain sub-folders. Assuming the following filesystem structure:

```
my-folder/
├── 1.txt
└── child_folder/
    └── 2.csv
```

where 1.txt contains

```
Obamacare is a great medical insurance!
```

and 2.csv contains

```
Header1, Header2, Header3, Header4
Huge, bank account, is also, "a great medical insurance."
```

Then the expected output will be:

```
financial
healthcare
```

GOOD LUCK!