Project Description

The project's goal is to build a **File Processing System**, where the system takes a **File Processing Scenario** as input, and then executes this scenario.

Conceptual Description

The scenario is given to the system as a text file with JSON format which is introduced in the "Scenario Description Format" section. However, from a conceptual point of view (i.e. ignoring the details of how the information is stored in the Scenario file), there are two main concepts in a scenario: **Entries** and **Processing Elements**.

Entries

Each entry refers to a file or directory. There are two types of entries:

- Local Entry: which is represented by a string value and refers to a file or a directory on the local file system. Some examples:
    - A local entry with the value 'c:\sample\txt\addresses.txt', refers to a single *file* on the local file system.
    - A local entry with the value 'c:\sample\txt', refers to a single *directory* on the local file system.
- Remote Entry: which refers to a file or a directory on a **Laserfiche Repository** on the cloud. Each remote entry is determined by a pair of values (repoId, entryId) where
    - *repoId* is the *id* of the Laserfiche Repository where the entry resides. This is a string with a maximum length of 20 characters, e.g. "*r-34w6"*
    - *entryId* is the *id* of the entry. This is an integer greater than 0. Note that simply by looking at the value of an *entryId*, we cannot say whether it belongs to a file or a directory.

Processing Elements

Each processing element is responsible for a unit of processing in the scenario. Each processing element takes a list of entries as input, performs some operation on them, and then generates a list of entries as output. A scenario includes a sequence of processing elements, so that the output of a processing element is the input of its successor processing element.

There are various types of processing elements:

- From a *functional* point of view, there are four types of processing elements:
    - *Filter*, *Split*, *List*, *Rename*, *Print*
- From a *storage* point of view, each of the above processing elements can have two versions:
    - *Local* Processing Elements: these are the processing elements that their input and output are all local entries. So, it performs the intended operation on local entries.
    - *Remote* Processing Elements: these are the processing elements that their input and output are all remote entries. So, it performs the intended operation on remote entries.

Next, various types of processing elements (from the functional point of view) are introduced.

Filter Processing Element

There are different types of Filters:

- **Name** Filter
  - Input: a list of entries, and a string value *Key*
  - What it does: For each entry in the input, it passes the entry to the output if its name contains the given string Key, otherwise, that entry is not included in the output.
  - Output: a sub-list of the entries with the given string Key in their name
- **Length** Filter
  - Input: a list of entries, a long value *Length*, and a string value *Operator*
  - What it does: For each entry in the input, if it is a file (not directory) and its length has a special relation with the given value Length, then it is passed to the output, otherwise, that entry is not included in the output. The relation is determined based on the given value of the Operator. The possible values for Operator are:
    - EQ: equal to
    - NEQ: not equal to
    - GT: greater than
    - GTE: greater than or equal to
    - LT: less than
    - LTE: less than or equal to
  - Output: a sub-list of the entries whose length satisfies the given Operator, with regard to the given Length.
  - Example: if a *Length* Filter is used with *Length=1000* and *Operator=GT*, it means the output of the filter is a sub-list of the input entries that are files and their length is greater than 1000 bytes.
- **Content** Filter
  - Input: a list of entries, and a string value *Key*
  - What it does: for each entry in the input, if it is a file and its content contains the string Key, then it is passed to the output, otherwise it is not included in the output.
  - Output: a sub-list of entries where the given string *Key* is present in the content of the entry
  - This filter treats the input files as text files and determines if the content of the file contains the given string.
    - For the purpose of simplicity, this filter reads the content of the file line by line and checks if there's a line which contains *Key*.
- **Count** Filter
  - Input: a list of entries, a string value *Key*, and an integer value Min greater than 0

- What it does: for each entry in the input, if it is a file and the string Key occurs at least Min times in the content of the file, then it is passed to the output, otherwise it is not included in the output.
- Output: a sub-list of entries where the given string Key appears at-least Min times in the content of the entry

Split Processing Element

- Input: a list of entries, and an integer value *Lines* greater than 0
- What it does: for each entry in the input, if it is a *file*, then it splits the file on a line by line basis, based on the given value *Lines*. The original *file* is kept unchanged. If the entry is a *directory*, it is ignored.
- Output: the list of created entries resulted from splitting the input entries.
- Example: Let's say the input includes 3 entries, where the first entry is a text file named "File1.txt'' with 60 lines, and the second entry is a text file named "File2.txt" with 25 lines, and the value of Length is 10. Then this processing element splits File1.txt into 6 files, each containing 10 consecutive lines from File1.txt. These files are named File1.part1.txt, File1.part2.txt, …, and File1.part6.txt. Further, File2.txt is split into 3 files, the first two including 10 lines, and the third one including 5 lines from File2.txt. These files are named File2.part1.txt, File2.part2.txt and File2.part3.txt. Finally, the output of the Split Processing Element is a list of the generated files, i.e. File1.part1.txt, File1.part2.txt, …, and File1.part6.txt, File2.part1.txt, File2.part2.txt and File2.part3.txt

List Processing Element

- Input: a list of entries, and an integer *Max* greater than 0
- What it does: for each entry in the input list, if it is a directory, a list of Max entries from the entries in that directory are passed to the output. If the number of entries in that directory is less than Max, all of them are passed to the output.
- Output: a list of entries selected from the directories in the input entry list.
- Example: Let's say the input includes 2 directory entries. Further, the first directory has 10 inner entries, and the second directory has 50 entries. If the value Max is 20, then the output includes all the 10 entries from the first directory, and 20 entries from the second directory (it is not important how those 20 entries are selected out of the existing 50 entries).

Rename Processing Element

- Input: a list of entries, and a string value *Suffix*
- What it does: for each entry in the input list, it appends the given string *Suffix* to the name of the entry. For instance, if an input entry has the name "file1.txt" and Suffix="_copy", then the name of the entry becomes "file1_copy.txt"
- Output: a list of entries, including the same entries as the input, with the updated names.

Print Processing Element

- Input: a list of entries
- What it does: prints the information of the entry. For local entries, this includes the name, length, and absolute path of the entry. For remote entries, this includes the entryId, name, length, and absolute path of the entry.
- Output: the same list of entries as input

Scenario Description Format

A file that contains a File Processing Scenario, is a JSON file with the following structure:

```
{
  "name":"Scenario Name",
  "processing_elements":[
    …
  ]
}
```

Where … replaces the description of the processing elements, separated by comma.

Each processing element has the following JSON format:

```
{
  “type”: “processing_element_type”
  “input_entries”: [
…
  ],
  “parameters”: [
..
  ]
}
```

Where the first … is a placeholder for the list of entries, and the second … is a placeholder for the list of parameters of that specific type of processing element.

Note: for the second and the later processing elements, the input_entries element is empty, because the input to each processing element is the output of the previous processing element, so it doesn’t need to be explicitly specified.

Further, each local entry has the following JSON format:

```
{
 “type”: “local”,
 “path”: “path to the entry”
}
```

And each remote entry has the following JSON format:

```
{
 “type”: “remte”,
 “repositoryId”: “repository id”,
 “entryId”: “entry id”
}
```

Further, each parameter has the following JSON format:

```
{
  “name”: “parameter name”,
  “value”: “parameter value”
}
```

Here’s the list of parameters for each type of processing element.

Processing Element Type	Parameter names
Name Filter	Key
Length Filter	Length, Operator
Content Filter	Key
Count Filter	Key, Min
Split	Lines
List	Max
Rename	Suffix
Print	

Sample File Processing Scenario

Below is a sample File Processing Scenario in JSON format. Here's the description of this scenario:

This scenario is named "First Scenario". It has 3 processing elements:

- The first processing element is of type List. And its input entries is a list of one local entry. This local entry refers to the directory in path "c:\sample\text_files". Further, the value of the parameter Max is 100. So, when this processing element is executed, it looks into the directory "c:\sample\text_files" and returns a list of a maximum 100 entries from this directory. This list of entries is passed to the second processing element.
- The second processing element is a *Length* Filter and the value of its parameters are Length=1024 and Operator="GTE". So, when this processing element is executed, it selects from its input entries those files that their length is greater than or equal to 1024 bytes. These files are passed as a list of entries to the third processing element.
- The third processing element is of type *Print*. So, when it is executed, it simply prints the information of the entries passed to it.

As a result, this scenario can be briefly described as: First, select a maximum of 100 files from the directory "c:\\sample\\text_files", and then print the information of those selected files that their length is greater than or equal to 1024 bytes.

```
{
 "name": "First Scenario",
 "processing_elements": [
  {
   "type": "List",
   "input_entries": [
    {
     "type": "local",
     "path": "c:\\sample\\text_files"
    }
   ],
   "parameters": [
    {
     "name": "Max",
     "value": "100"
    }
   ]
  },
  {
   "type": "LengthFilter",
   "input_entries": [],
   "parameters": [
    {
     "name": "Length",
     "value": "1024"
    },
```

```
      {
        "name": "Operator",
        "value": "GTE"
      }
    ]
   },
   {
    "type": "Print",
    "input_entries": [],
    "parameters": []
   }
  ]
}
```

What you need to do

Briefly stated, you need to develop the required hierarchy of classes for implementing various types of entries and processing elements mentioned above. Further, you need to develop a program that

- takes a File Processing Scenario file as input
- parses the scenario file and creates the required objects from the corresponding classes
- Executes the scenario by giving the required inputs to the first processing element, and passing along the output to the next processing element, up to the last processing element.

As mentioned before, there are two main type of entries:

- those residing in the local file system. To access/process these entries you can use Java file processing api, (e.g. File and BufferedReader classes)
- those residing in a Laserfiche repository. To access/process these entries which are actually on the cloud, you need to use the so-called Laserfiche Client Library which can be found on GitHub at https://github.com/Laserfiche/lf-repository-api-client-java. Further information on how to use this client library **will be provided in a separate document**.

Additional points on processing elements

- The operation performed by some processing elements involves string comparison. For instance, a Name Filter needs to compare the name of the entries with a given string. Wherever string comparison is needed, ignore case-sensitivity, i.e. compare in a case-insensitive mode.
- The operation performed by some processing elements involves getting the length of an entry. If an entry is a directory, consider its length to be 0. Otherwise, it's length needs to be looked up.
- Note that the order of execution of some processing elements is important. For instance, having two processing elements P1 and P2, correspondingly of type *List* and *Length Filter*, the final result of executing [P1, P2] is not necessarily the same as of [P2 -> P1].