**Project Name**: ReadProcessWebData

**Jar name**: ReadProcessWebData.jar

**Jar path**: \ReadProcessWebData\out\artifacts\ReadProcessWebData\_jar\

**Run jar file this way**: enter command line, go to jar path, run: java -jar ReadProcessWebData.jar

Jar file will execute a program code that perform the given task.

Given task was to read search results content from web site GitHub and then to write the content in files. The search results content will spread between 50 files.

Was requested to perform the task in 2 steps:

**Step\_1**: read all the web content into single JSON file – use selenium web driver

**Step\_2**: Spread the content of the single file into multiple json files efficiently

**Step\_1** took **2.5** **days**. During it I learned and made hands on implementing it.

**Step\_2** took **1.5** **days**. During it I learned and read about thread pool pattern and made hands on implementing it.

**Step\_1:**

**Learn and Designing**: (**1 d**)

I learned about what is **selenium**, its components and each component role and capabilities,

How to connect WebDriver and brows to web site.

How to get specific WebElement on page.

I used mostly the web tutorials and youtube.

I also decided to perform data collecting and data processing in different files and classes from the logic aspects.

**Implementing** : (**1.5 d**)

I got required jar file and implemented a connection of webDriver.

I succeeded to connect to brows to GitHub site, to get search element on the main git hub site, to get all the pages with the search results. Per result, I got title, description (if exists) but importunately had difficulties to get xpath to the rest of the search result elements (tags, stars, language).

I have made a decision to **simulate** web data and keep working with simulated search results. While I concentrating on finishing all the projects parts.

So in Step 1, I generated simulated search results and stored them in Jason file. While the naming convention of the file is preserved as defined by task, for exp generated file with simulated search results is: **SecurityResultGitHub-14-09-2020\_12\_11\_07.0803.json**

Implementation of search results collection (simulation and storage) is done in separate java file: **SearchResultsCollector.java** and class.

Flow chart:

Web site

JSONArray

File

**SecurityResultGitHub-14-09-2020\_12\_11\_07.0803.json**

**Step\_2:**

**Learn and Designing**: (**1 d**)

I learned about what is **thread pool**, the advantages of using it mostly when we have many short common tasks /client requests to be treated, we should not create many threads as amount of tasks but to manage the process to store tasks in queue and to have a pool of threads that will be triggered by executer. Upon a need to perform new task from queue executer finds available thread and run it, this way the overall time to serve & overhead is lower. A size of the pool thread = amount of CPU cores of the machine.

**Implementing**: (**1.5 d**)

After a need & concept were clear to me I designed class architecture of this.

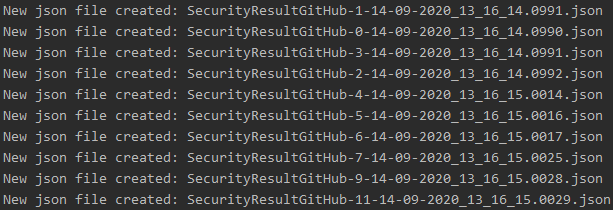
Processing and spreading search results implemented in separate file. I created new java file **ContentSpreader.java**

Here are two classes. Class **ContentSpreader** that implements the thread pool by running in loop over entire set of search results (JSONArray) that was previously prepared in step\_1 where all search results read from a file into JASONArray and per entry calling the executer to process the data in the entry using one of the available threads in thread pool.

Class **WorkerThread** - this is actually the runnable object. In its run() method, it actually it is doing:

* Creates file according to the naming convention: uses the index of the entry and adding it a creation time.
* Writing the received search into file.

For exp these are files that were created:



Flow chart:

File

**SecurityResultGitHub-14-09-2020\_12\_11\_07.0803.json**

Iterate it

JSONFile 3

WorkerThread

JSONFile 2

WorkerThread

WorkerThread

JSONFile 1

JSONFile 0

WorkerThread

JSONObject

JSONObject

JSONObject

JSONObject

JSONArray