COMP 1405Z – Fall 2023
# Course Project: Analysis Report
Kassem Taha & Barnabé Frimaudeau

## Instructions

The controller class is called "Controller.java". To run the GUI, you will need to run the "GUI.java" file.

## Functionality

| *Functionality* | *Complete / Incomplete* |
|---|---|
| Scrape through all interlinked web pages | **COMPLETE** |
| Store necessary data in directories and files | **COMPLETE** |
| Read all the necessary data in a reasonable amount of time | **COMPLETE** |
| Correctly calculate the search results asked by the user and sort them in a reasonable amount of time | **COMPLETE** |
| Display all of the results using a neat GUI | **COMPLETE** |

## Classes Outline

### Classes

#### osutil
This is our file handling class. It helps us create, read, and append files without having to worry about every detail. It helps with code readability, efficiency, and reuse.

#### crawler
The crawler class does most of the analyzing and computation in our program. It is responsible for gathering all the data we need for searching.

## ProjectTesterImp/Searcher

This class utilizes all the crawling data and is our head of operations. In this class, we read the files, calculate search score, and initialize everything in the program.

## SearchResult

This is a custom data type we made to house our SearchResult. It only has two main properties: title, and score which we use to output and sort.

## Controller

This class is part of the Model-View-Controller architecture pattern and it represents the controller component. Its responsibility is to handle the logic between the graphical user interface and the model (crawlMain & SearchResult).

## GUI

This class is also part of the MVC architecture, but this one represents the view component. Its responsibility is to display information to the screen in the form of a window. It takes in String input (crawl seed and search query), user button clicks (crawl and search), and a boolean checkbox for the usage of PageRank boost. It then displays the search results in a vertical list view, which lists the top 10 results from most to least relevant.

## Others

## WebRequester

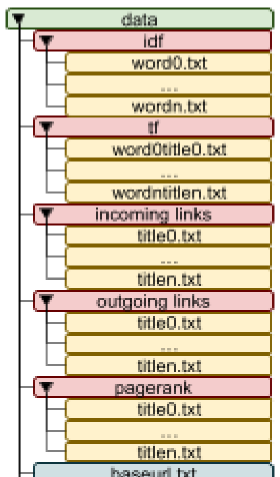
This is the base class given to us to acquire the string we parsed in crawler.

## ProjectTester

The interface provided to us which ProjectTesterImp/Searcher implements.

## Overall Design

During the programming phase of this iteration of the project, our primary focus was on transitioning our existing codebase. As a result, our class structure closely resembles that of the 1405 version. This similarity is particularly evident in our file structure, which remains largely unchanged, with minimal changes made to naming conventions. We opted to keep our previous file structure as it best suited our requirements and was the least complex arrangement we had in mind.



The osutil class was an important abstraction we developed. Given the difficulty involved in file manipulation, such as adding, deleting, and appending files, we found it counterproductive to directly interact with files at all times. Hence, the osutil class was the initial class we created from scratch to help with the next classes like crawler.

Equally essential is the crawler class, which parses and organizes data before handing it over to the osutil class for storage. The crawler performs a lot of computations and requires many variables, prompting us to make it as its own class. Through encapsulation, we do not worry about the larger number of lists, arrays, dictionaries, and other variables generated during its processes. This encapsulation prevents the need to worry about ten new variables within our main class, enhancing code efficiency.

This leads us to our "main" class Searcher/ProjectTesterImp. In this class, we capitalized on inheritance and implemented the provided ProjectTester interface. By doing so, we created a "main class" that effectively utilizes both the crawler and osutil to initiate the crawling process and subsequently read the obtained files. The ProjectTester interface proves beneficial as it amalgamates the functionalities of search and searchdata from the 1405 version.

With OOP, everytime I want to make the crawler parse more data from other HTML tags I simply just alter the crawler function without having to worry about the other classes. If I want to calculate the IDF*TF*Search Score I do not need to alter the crawler class, I can simply read the data we already have.

In conclusion, our approach to this project centered on using the working structure from our previous iteration (1405 version) while making necessary adaptations to use new concepts like OOP and create new features like the GUI. By applying modularity through class structures like the osutil and crawler classes, we achieved significant improvements in code organization, readability, and efficiency.