Dependency Inversion Principle

For our project, we have many different functional modules, which are tightly connected and dependent to each other. Instead of directly letting modules dependent to each other, we follow the Dependency Inversion Principle to make modules depend on abstractions and interfaces, thus decouple the coupling classes. In this way, if we change the implementation or create new module to replace the old one, we do not have to change the code of its dependents. This help to improve the ability to maintain and further improvement.

For example, in our project, we need generate two hash map data structures, web page URL mapping to all its keywords and each keyword mapping to URLs of all web pages containing this keyword. The modules to generate these two maps are highly coupled and depend on each other, which may make further code testing and maintainence difficult. Instead, we design that these two modules should depend on interfaces. This make these two components easier to maintain and improve.

Single Responsibility Principle

For our project, instead of making a large module responsible to different functionality, we split the module such that each module should act single responsibility. This help us make further improvement and refactor code. In contrast, for large module with multiple responsibilities, minor change on existing code may lead to problems in other dependent class.

For example, in our pre-process package, we split the pre-process functionality to extracting information from web page, generating hashmap from web page and generating inverse hashmap. By splitting the functionality to subclass, we can easily improve efficiency and performance of single method in a subclass.

Open-close principle

In preprocess module, we have string map module to generate a mapping from url to web page content, which will need extractor module to extract webpage content. Instead of directly connecting single extractor and string map module. We use abstraction of extractor to connect to the string map module. In this way, we can extend other extractors, which follows ‘open for extension’. And we can easily change code of string map module and extractor without changing their dependent class, which follows ‘ close for modification’.

For example, in our project, we need to extract different types of information in webpage, such as URL, keywords, webpage title.For example, in our project, we need to extract different types of information in webpage, such as URL, keywords, webpage title. With implementation of different extractors, we do not need to modify existing code of their dependent class.

Liskov’s Substitution Principle

As an extension of open-close principle, Liskov’s Substitution Principle states subclass must not replace its super class. In our project, for query module, we implement two different algorithm for sorting the query result, TF-IDF and Page rank. Both of them extend an abstraction class and implement different weight method.In this case, substituting the abstraction class with any of these two subclass will keep the functionality same.