UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

SOFTWARE ENGINEERING ASSIGNMENT 2 REPORT

REPORT
STUDENTMAN SOFTWARE
FINAL VERSION

Name	Vũ Quỳnh Anh
ID	BI12 - 019
Class	Data Science

1. Two interfaces Comparable and Document

There are two interfaces that we use in our program: **Comparable** and **Document**

- The **Comparable** interface is used to determine the ordering of the objects; the order is transitive and symmetric, and it is a total order over the objects. This interface only has one method, *compareTo(Object).*
- The keyword search engine reads the objects as HTML documents using the second interface, **Document**. The *toHtmlDoc()* method of this interface produces a String with the text of a basic HTML document that was constructed from the state of the current object.

a) Without using **Comparable**

- The Comparator interface may be used to create the same functionality without using **Comparable**. To set a custom sorting criteria for the objects, you can construct a new class that implements the *compare()* function. It also lets you sort objects using different criteria.

b) Without using **Document**

- Without using **Document**, we can directly write the method *toHtmlDoc()* for a class instead of building the **Document** interface.

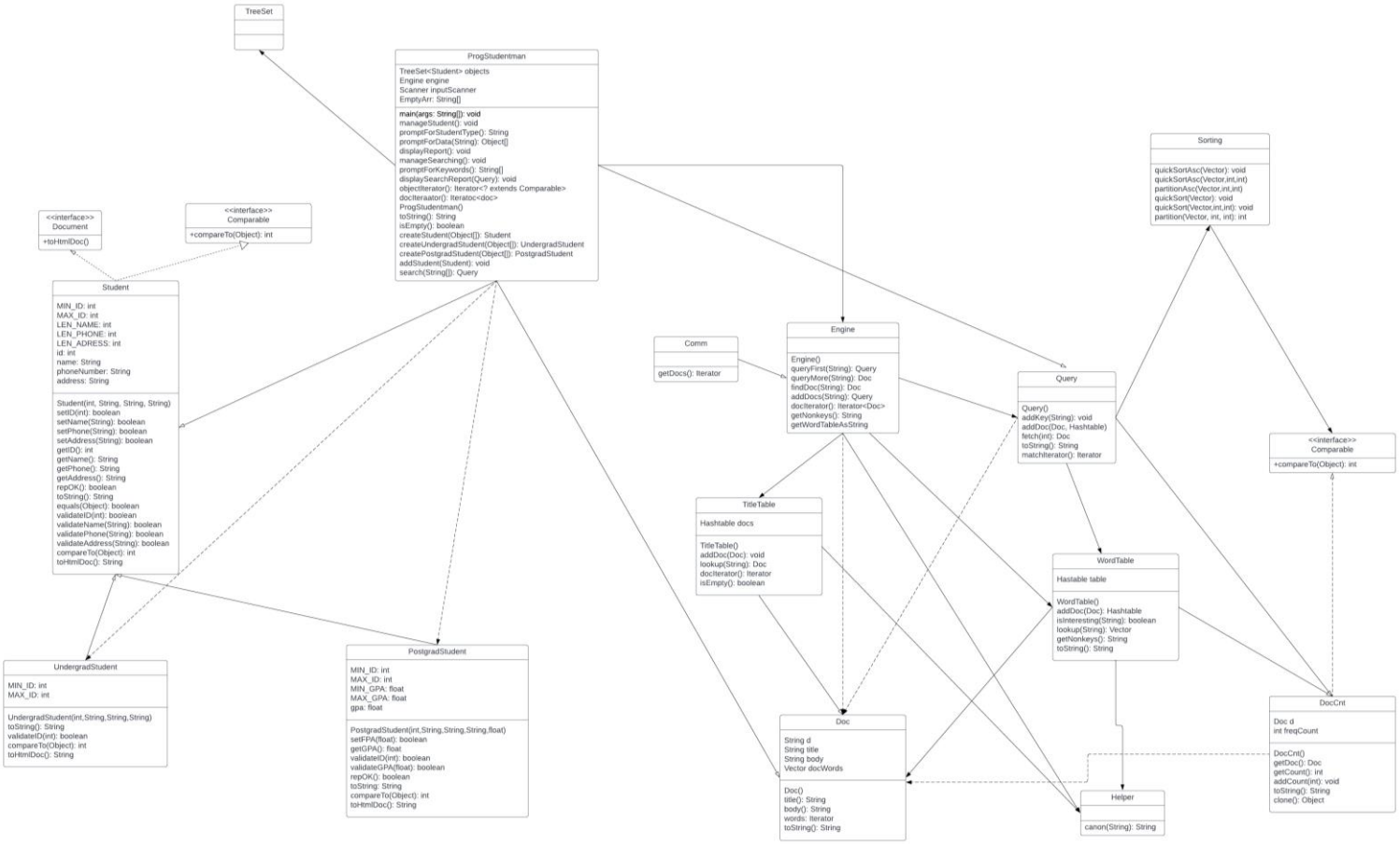
2. TreeSet Class

- The **TreeSet** class is used to create an empty **TreeSet** object in a sorted order. **TreeSet** is extremely handy for keeping a collection of unique items in sorted order and performing actions on them effectively.
- The same software functionality may be created without the use of TreeSet. To store the items, we may build an *ArrayList* and then use *Collections.sort()* to sort the *ArrayList* in the correct order. Finally, cycle over the sorted *ArrayList* and add each unique object to a new *ArrayList* using a loop. While iterating over the sorted list, you may utilise a hash set to check for duplicates.

3. KEngine library

The original KEngine library can only use keywords to search for text documents. We may, however, search for Student objects using keywords by using *toHtmlDoc().* This function will return a string of text from an HTML document built from the current object's state. The Kengine use this approach to build an HTML document, which it then searches for the specified keywords within.

4. UML



5. Implementation strategy

In order to reduce slag time, we employ hybrid implementation for our program, which combines top-down and bottom-up approaches. We start with the **Student** class and then go on to its two subclasses, **UndergradStudent** and **PostgradStudent**. Without stubs, these three classes will be incomplete. Next, we create the **Document** interface, which the **Student** class will extend. We now add the *toHtmlDoc()* function to finish the **Student** class and its two subclasses. The **ProgStudentman** class will only be partially implemented with stubs, and we finish the **Engine** and **Query** classes by implementing the *addDoc()* and *matchIterator()* methods, respectively. Finally, by completing all of **ProgStudentman's** methods, we complete our software.