Assignment 2: StudentMan Software - Final Version

In this assignment, you will work individually to build a moderately complex software that captures data about domain objects and allows the user to search for objects of interest by keywords. This software will use library components from the previous assignments and from the lectures.

1. Requirement

Program StudentMan, which was introduced earlier in the semester, now needs to be upgraded in order to turn it into a software more suitable for use by its users.

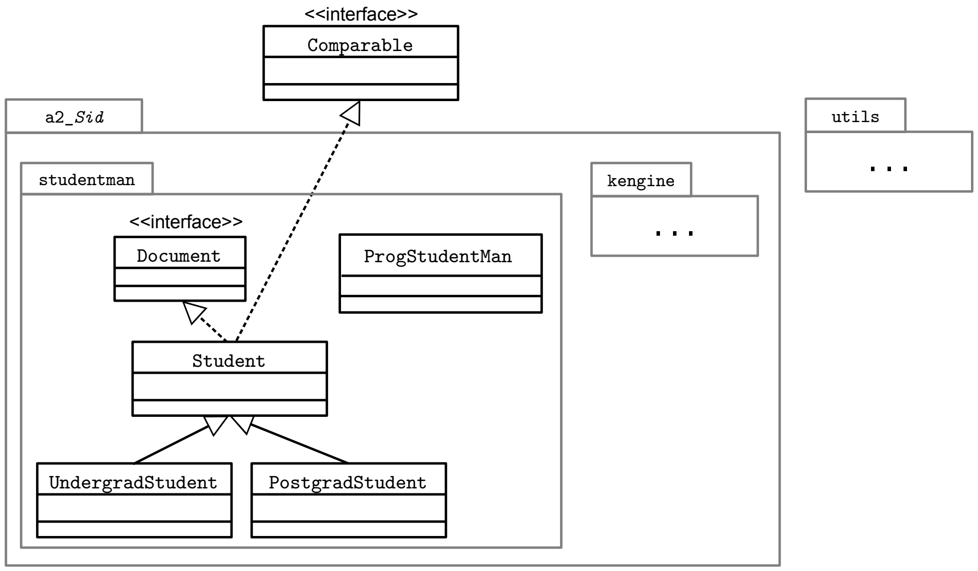

Figure 1: A high-level UML class diagram of the software.

Figure 1 is the high-level UML class diagram of the software. The upgrade tasks that you need to perform are described in Section 2. You must use the classes that you developed earlier in the semester for program StudentMan to carry out the upgrade. The success of StudentMan software depends on these classes working correctly. So **if you have not satisfactorily completed these classes** then you must do so before commencing work on this assignment.

2. Tasks

1. Create a top-level package named `a2_`*`Sid`* as shown in Figure 1, where *`Sid`* is your student id. For example, if your student id is 123456789 then the package name is `a2_123456789`. Next, create two sub-packages: `studentman` and `kengine`, by copying them from the attached assignment zip file. Create another top-level package named `utils`, also by copying it from the attached assignment zip file. This library includes necessary utility classes needed to design the software. Package `kengine` includes the KENGINE library component, while package `studentman` contains all the classes that you specifically develop for the StudentMan software.

   Initially, copy into package `studentman` the classes for StudentMan that you developed earlier in the semester or have revised. In the tasks that follow, you will need to modify some of these classes and also create some other classes in the package `studentman`. In addition, you will modify two classes `kengine.Engine` and `kengine.Query` a little.

IMPORTANT: Package `utils` *must not* be created as a sub-package of package `a2_`*`Sid`*. Failure to create the packages as described above will result in an invalid software.

2. In the package `studentman`, specify an interface named `Document` that will be implemented by class `Student` so that its objects can be read as HTML documents by the keyword search engine. This interface contains a single method named `toHtmlDoc`, which takes no argument and returns a `String` containing the text of a simple HTML document generated from the state of the current object. This document must follow the format described in the following example.

   Suppose the above operation is invoked on the `Student` object `Student:<4, John, 12345678, Hanoi>`, then the output is:

```
<html>
<head><title>Student:4-John</title></head>
<body>
4 John 12345678 Hanoi
</body></html>
```

   The output formats for `UndergradStudent` and `PostgradStudent` objects are the same, except for the case of `PostgradStudent` where the extra `gpa` value is added to the content line of the body.
3. Update the classes of the `Student` type hierarchy to appropriately implement `Document` interface.
4. Specify and implement method `Query.matchIterator()` which returns an `Iterator` of the query matches or `null` if no matches. You will need this method to generate the HTML search report.
5. Implement method `Engine.addDoc` that follows the design specification given below. This is the only modification to this class that you need.

```
/**
 * @effects
 *   if d is null
 *     throws NullPointerException
 *   else
 *     add d to this.tt and this.wt using their respective methods.
 *     If this.q is not null
 *       update this.q to contain any new matching documents.
 *     Return this.q
 */
public Query addDoc(Doc d) throws NullPointerException
```

6. Complete the code of the class `ProgStudentMan`. The specification and partial code of the this class is given in the attached file named `ProgStudentMan_spec.txt`. The code that you need to write for this class is clearly marked in the file. Note the followings:

   6.1. change the package declaration and some import statements that currently refer to the top package named “`a2`” to your `a2_`*`Sid`* package

6.2. strictly follow the specification given. DO NOT modify it or create your own specification

6.3. you must not change the partial code that is given

7. Write a ***brief*** design note document to give details of your answers to the following questions. Use the bullet-point form and table where possible.

7.1. What is the purpose of using the two interfaces `Comparable` and `Document`?

Is there another way of designing the same functionality (a) without using `Comparable` and (b) without using `Document`? Briefly explain why or why not under each case?

7.2. What is the purpose of using the `TreeSet` class in the software? Is it possible to develop the same software functionality without using this class? Briefly explain why or why not?

7.3. The original KENGINE library can only search for text documents using keywords. What makes it possible to use this component in your software to search for `Student` objects using keywords?

7.4. Draw a complete UML design class diagram of the software showing the relevant classes (with attributes and operations) and their dependencies.

7.5. Based on the design class diagram and the previous tasks, identify the implementation strategy that was used to build the software. Briefly discuss this strategy.

3. Submission

You must submit the code and the design note document using separate submission boxes as follow:

- Create a **zip-compressed file** containing **just the folder of the top-level package** named `a2_`*`Sid`* (described in Task 1). You must name the file as follows: `a2_`*`Sid`*`.zip`, where *`Sid`* is your student id. Submit this zip file to the designated submission box for this assignment.
- Name the design document `a2_`*`Sid`*`-design-note`, where *`Sid`* is your student id.

  This document must have a standard cover page that states the assignment title and the student details. In addition, it must be submitted in the Adobe Acrobat (**.pdf**) format.

  Submit this file to the submission box named "Assignment 2 - Report".

IMPORTANT: Failure to prepare the files as described above will result in an invalid program. In particular, ONLY the **ZIP** format is accepted.