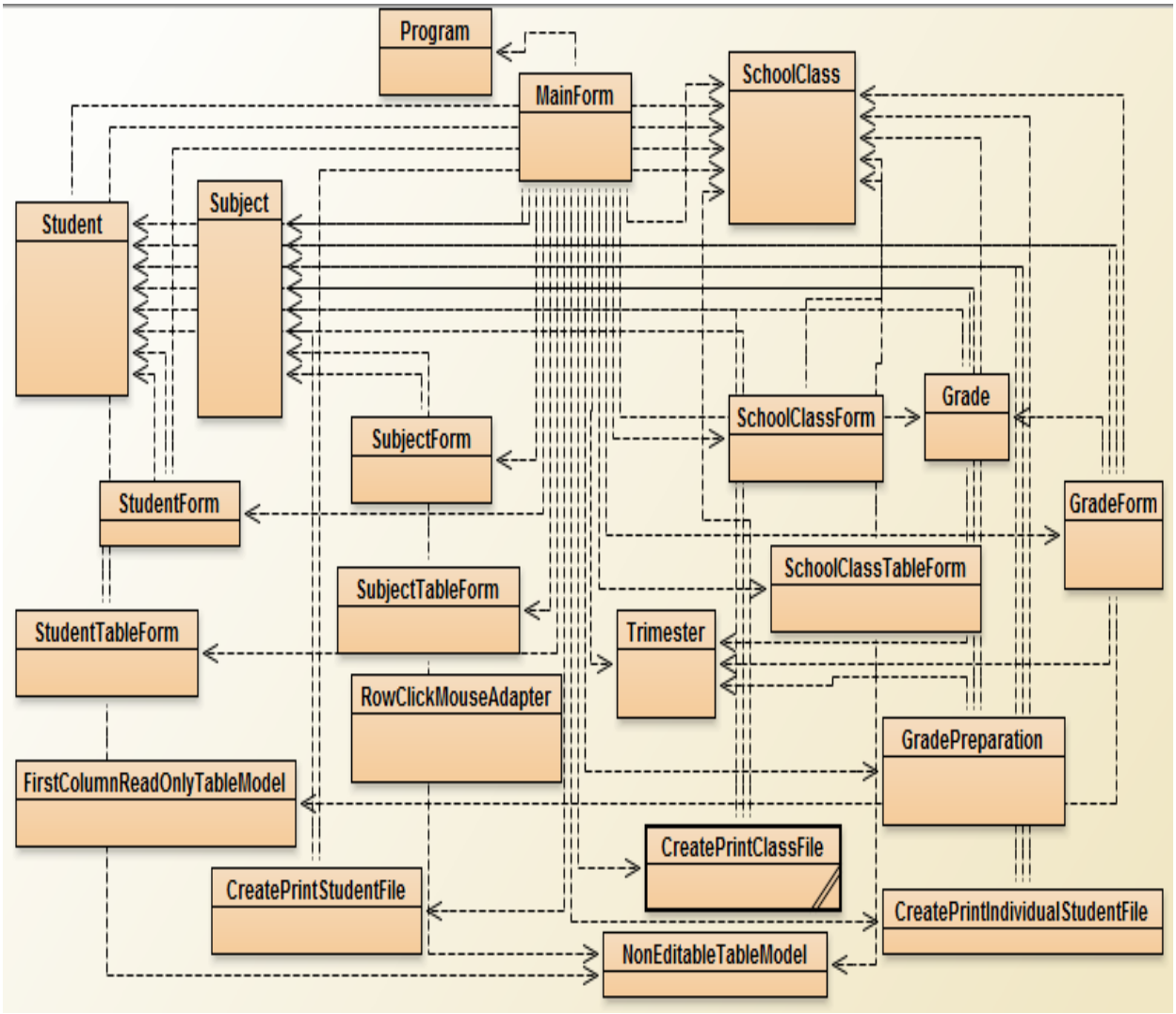
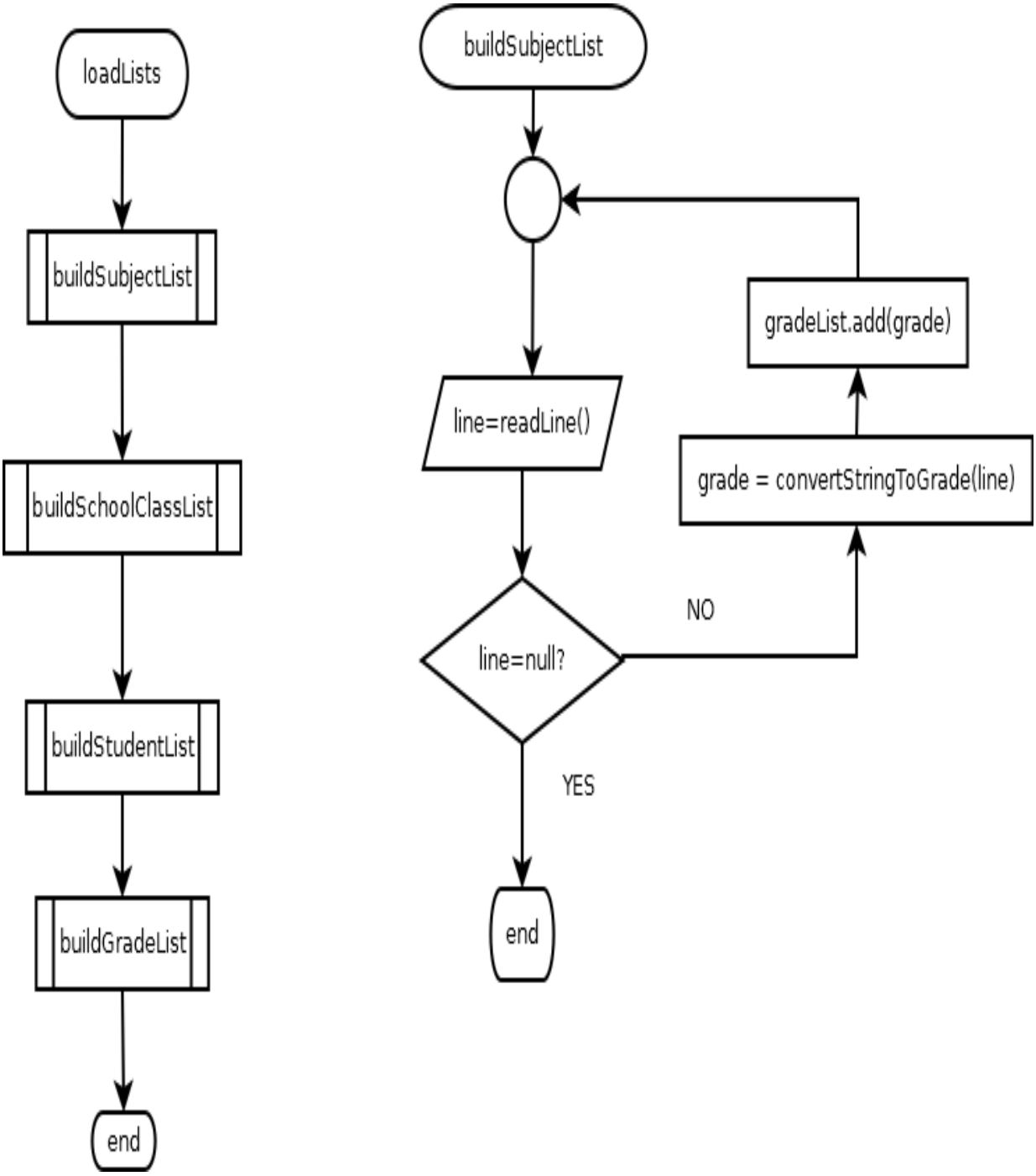
Criterion C: Development

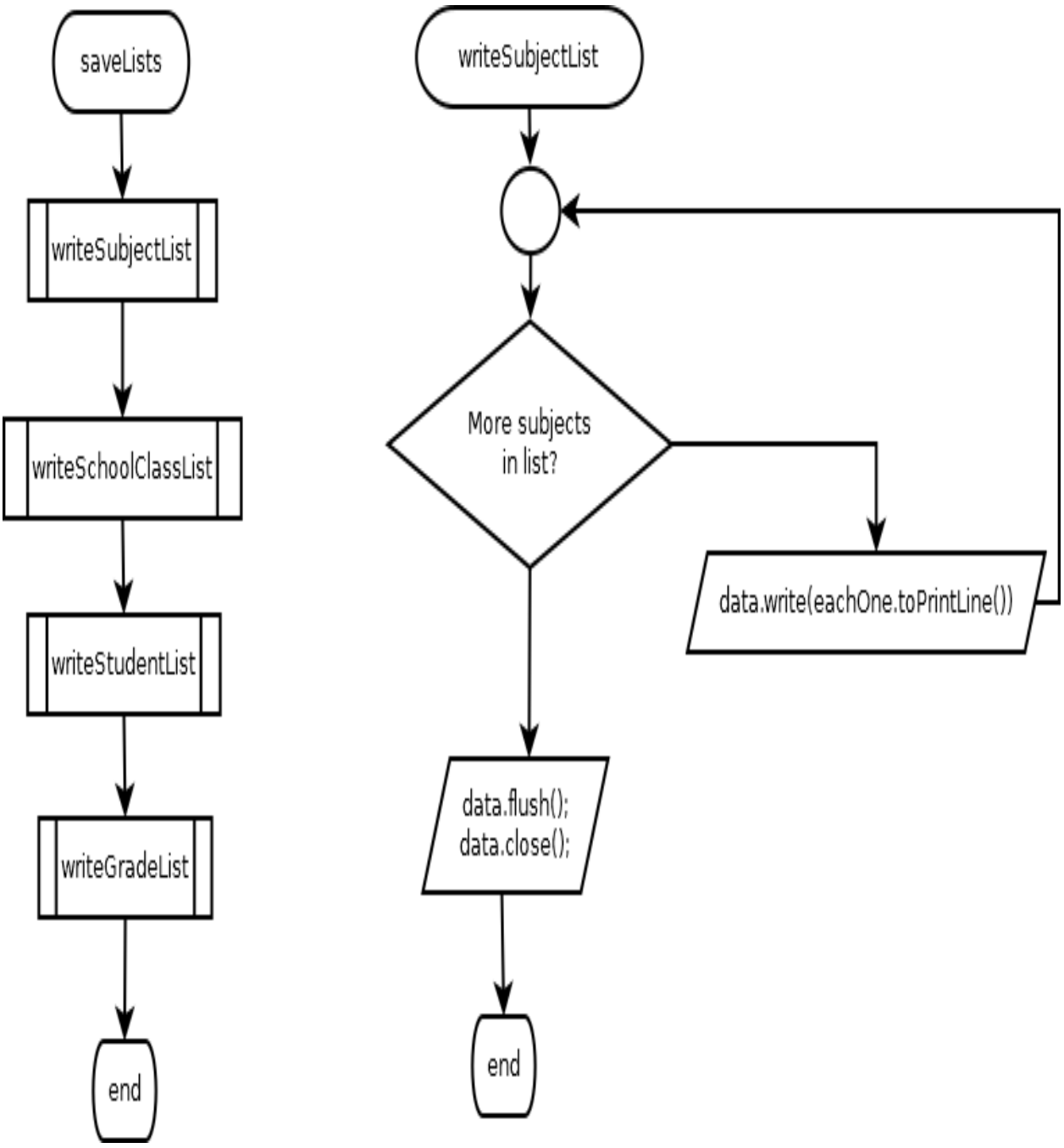


(Picture 1.1 – program structure and class dependency)

The program always starts and finishes in the same way; in the beginning, the data contained in the files are loaded into the sets and at the end, all data are stored into new files that overwrite the previous ones, and are stored on the hard disk. This process can be seen in the two diagrams below:



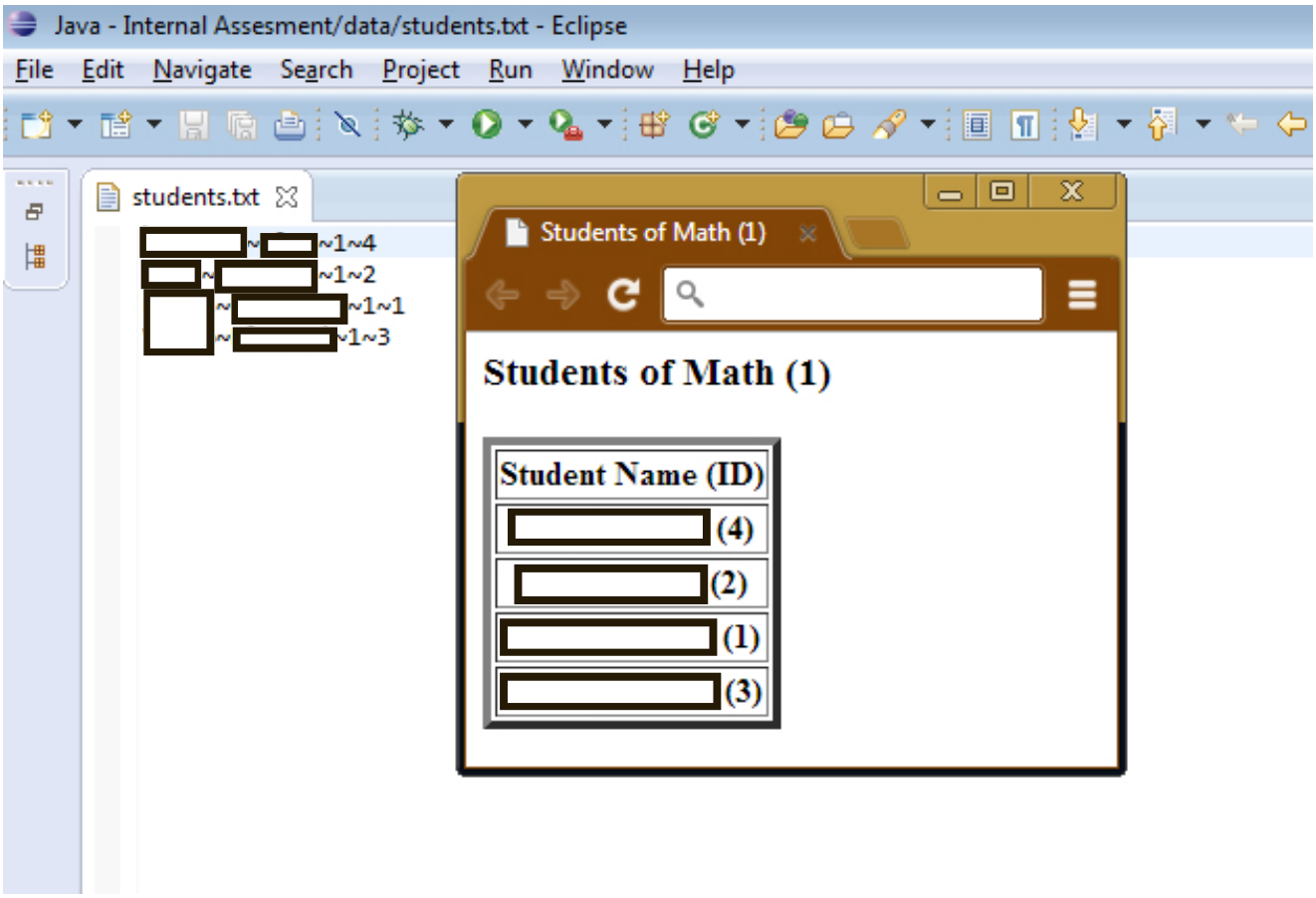
(Picture 2.1 – loadLists flowchart)



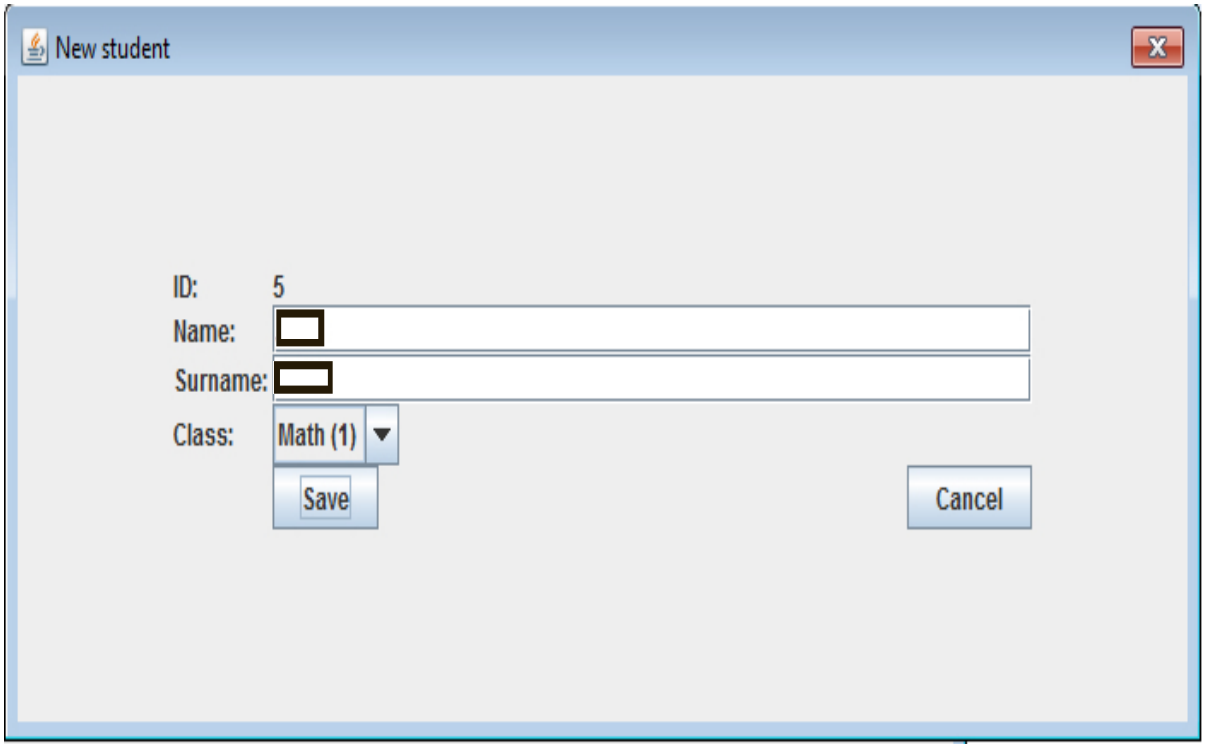
(picture 2.2 – saveLists flowchart)

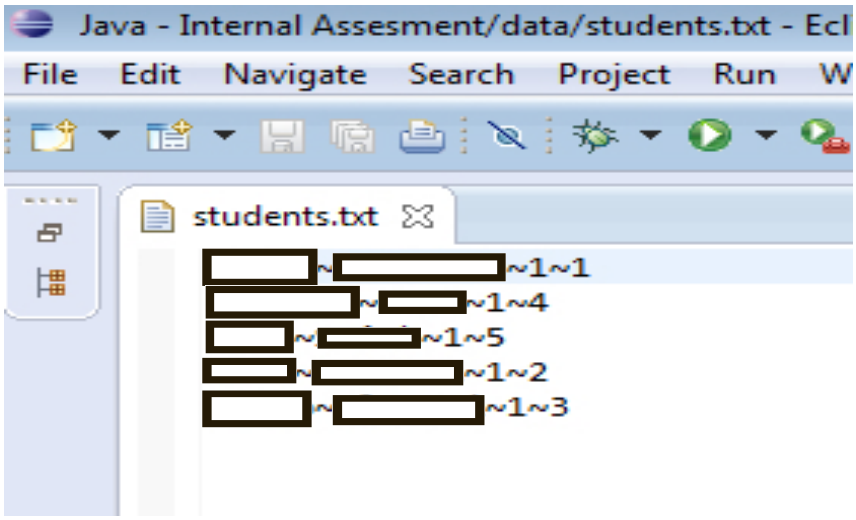
Note: the methods not described in the flowcharts follow a similar pattern as the ones that are included.

In order to test if these operations work correctly, I designed two testing scenarios: for the loadLists, data will be stored manually on the files and then printed, and for the saveLists, through the program, new data will be stored in the sets and after the program is closed, the files will be checked for the result.



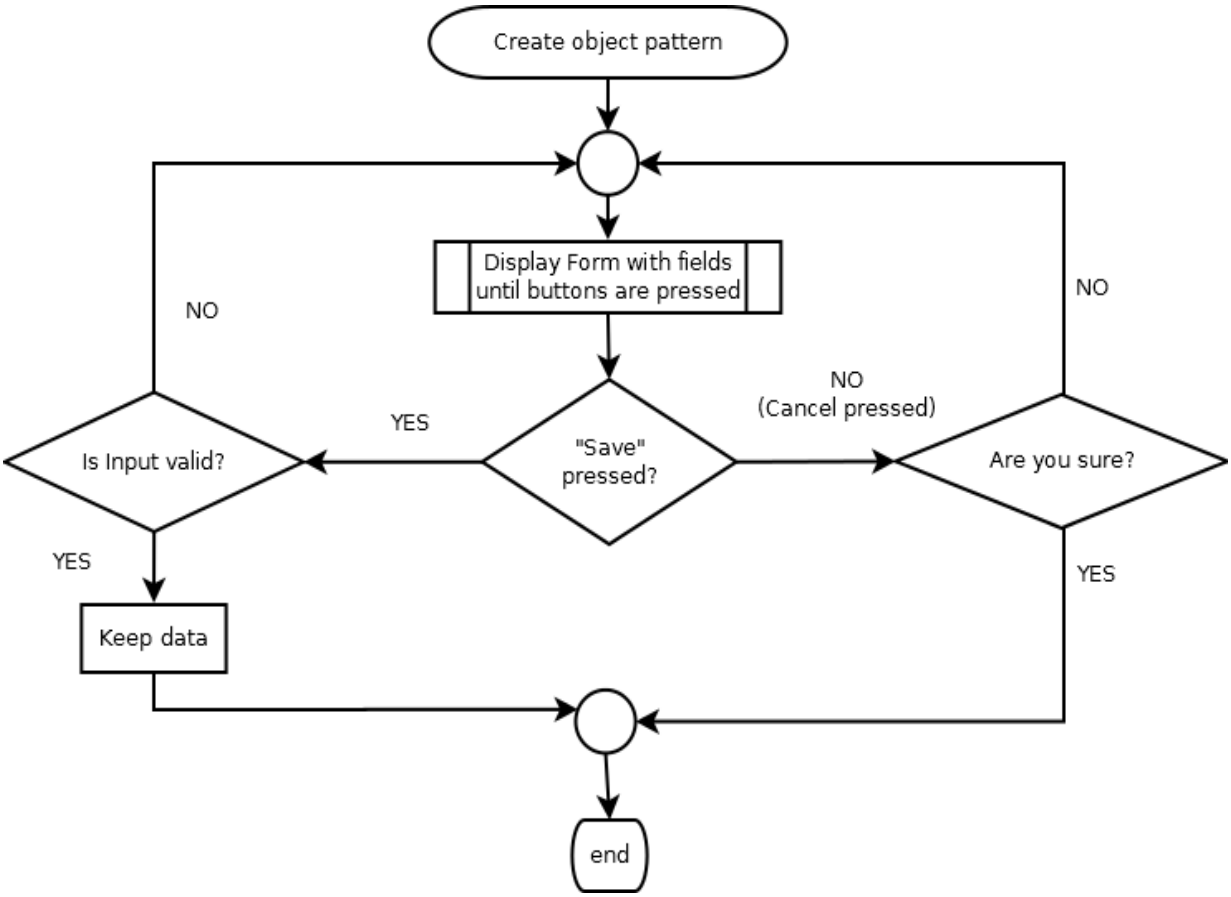
(Picture 2.3 - loadLists testing example)





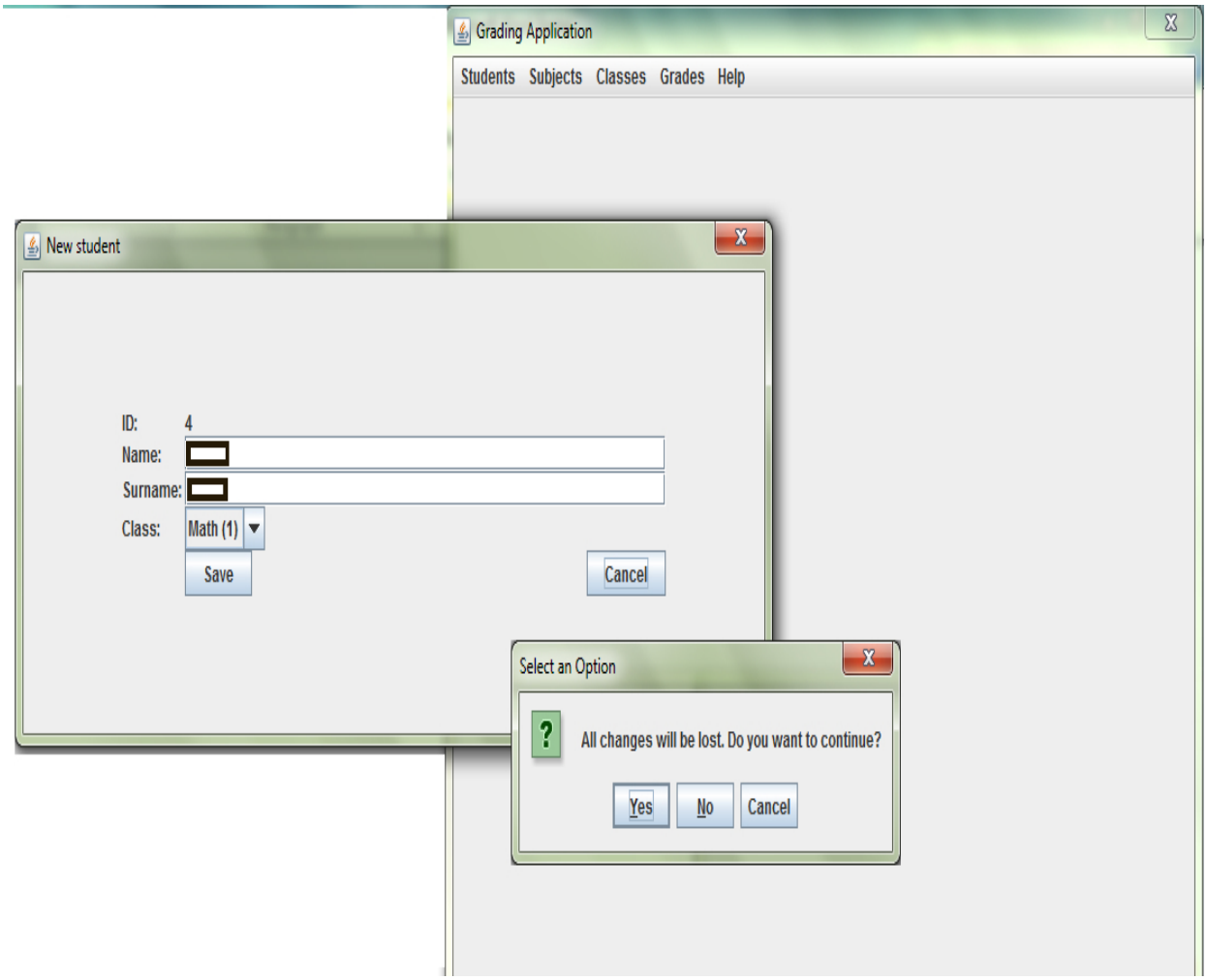
(Pictures 2.4 & 2.5 – saveLists testing example)

Furthermore, in an effort to have uniformity, as mentioned in the design section, and as a result of the Single Responsiility Principle that was used, all forms follow a similar pattern. For example, almost every form has a save and a cancel button that perform similar actions:



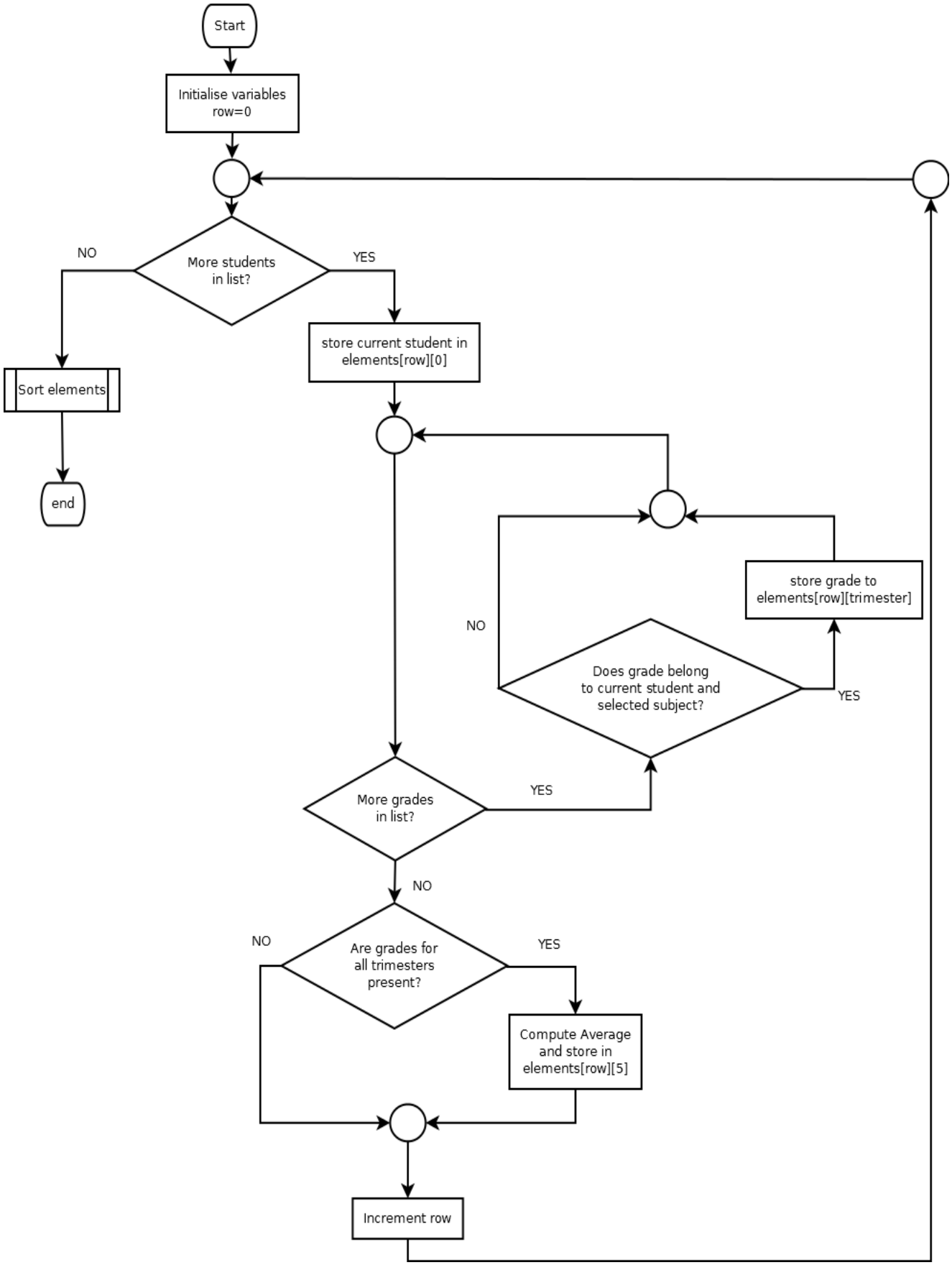
(Picture 3.1 – save and cancel buttons)

An example of the confirmation questions that are used can also be seen in the screenshot that follows;



(Picture 3.2 – confirmation question)

Another important feature of the application is that the user can print student lists, individual and collective student records, using a browser. This depends heavily on the fact that every object has a unique ID, which acts as a surrogate key and it differentiates it from the rest. However, there are cases where a composite primary key is used, because the object is always identified using multiple fields. An example of a composite key object can be seen in the following print command:



(picture 4.1 – printClassFile flowchart)

Finally, a Java sorting algorithm was used, because after studying the following oracle documentation, I decided that it covered fully my needs and that as an app programmer, I did not have to worry about creating an algorithm that had been already created.

*"The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist). This algorithm offers guaranteed n*log(n) performance."*

(source: http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html)

word count: 407