# Assignment 4

Implement the next release of your term project or start a new one if necessary. You will incorporate the *saving and retrieval of objects*, the use of *lambdas*, and the use of *streams*. You can substitute the use of JavaFX for one of these three if you wish; substitutions for any of these concepts by advanced techniques are acceptable if you have already used them and obtain agreement from your facilitator.

The same instructions as in Assignment 3 apply to this completed Word document regarding the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 4.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 2.

Knowla (Knowledge assessment and learning) is a scrambling/unscrambling tool for instructors and students to learn material. Knowla facilitates testing on the material and being automatically graded, which is the focus of this project.

This project concerns the grading of Knowla tests, which we will call *K-Grade*. K-Grade allows the instructor to assign points to qualities that they want in students answers—expressed in terms of rubric sets. The grading philosophy is to assign only positive points (i.e., not to subtract for deficiencies). The instructor is prompted to enter a sequence of student names and answers.

## 4.2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear which are new vs. old.

### 4.2.1 Entry of Student Answer (NEW)

### The instructor is prompted to enter a sequence of student names and answers.

### 4.2.2 Checking (NEW)

### Student answers are checked for maximum and minimum. Only numbers bounded by these are accepted, and are reported to console—without spaces.

## 4.3 I/O SUPPORTING THE REQUIREMENTS LISTED ABOVE

Provide an example of input / output showing the new features of your application.

### Input

File order-rubrics.txt

4 6 8 You are correct that #4 occurs before #6.

2 4 5 You are correct that #2 occurs before #4.

1 2 7 You are correct that #1 occurs before #2.

File consecutive-rubrics.txt

1 2 7 You are correct that #1 and #2 are consecutive.

5 6 2 You are correct that #5 and #6 are consecutive.

3 4 8 You are correct that #3 and #4 are consecutive.

### Input / Output

Console I/O:

enter 'N' if no more students--otherwise, student's name: Eric

You input: Eric

Please enter student answer as in '1 2 3 6 8 4': 1 2 3 6 8 4

Student answer is--->1 2 3 6 8 4

Sorted entries within bounds (not blank-separated):

123468

Enter N if no more students--otherwise, student's name: June

You input: June

Please enter student answer as in '1 2 3 6 8 4': 2 4 6 8 10

Student answer is--->2 4 6 8 10

Sorted entries within bounds (not blank-separated):

246810

Enter N if no more students--otherwise, student's name: N

You input: N

EOF reached in student-ungraded-repository.dat

Grading complete; records stored as StudentPaper objects in student-graded-repository.dat

======THE TOTAL NUMBER OF POINTS IS======>37.0

==>Eric: Answer ...[1, 2, 3, 6, 8, 4] Grade ...19.0

Feedback:

You are correct that #1 and #2 are consecutive.

You are correct that #2 occurs before #4.

You are correct that #1 occurs before #2.

==>June: Answer ...[2, 4, 6, 8, 10] Grade ...13.0

Feedback:

You are correct that #4 occurs before #6.

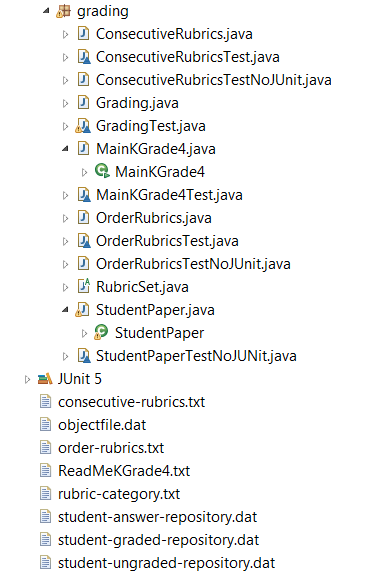
You are correct that #2 occurs before #4.

EOF reached in student-ungraded-repository.dat

==========ALL STUDENTS REPORTED===========

## 4.4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written—more on this later). This should include JUnit tests—class-by-class, and method-by-method, except for trivial ones.



## 4.5 DESIGN

### 4.5.1 Class Model, Use Case, and Sequence Diagram

## Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas.

Please see [here](https://docs.google.com/spreadsheets/d/1a12k2TX75jGhvsnYTkMe5-ZM3jqAmujP0DI54kHH_48/edit?usp=sharing).

### 4.5.2 Code showing object read and write

.

There are a few places where this is done. The following is from *MainKGrade.gradeStudentAnswers*().

CODE:

**public** **static** **void** gradeStudentAnswers() {

/\*

\* Precondition: StudentPaper objects, including names and answers, are in

\* STANDARD\_UNGRADED\_STUDENT\_PAPERS

\* Postcondition 1: StudentPaper objects, including paperGrade corresponding to

\* gradeWithAllRubrics(), are in STANDARD\_GRADED\_STUDENT\_PAPERS

\* Post2 (Acknowledged): Grading was acknowledged for each paper, by name.

\*/

**try**

{

**try** (ObjectInputStream infile = **new** ObjectInputStream

(**new** FileInputStream(***STANDARD\_UNGRADED\_STUDENT\_PAPERS***));)

{

**try** (ObjectOutputStream studentFile = **new** ObjectOutputStream

(**new** FileOutputStream(***STANDARD\_GRADED\_STUDENT\_PAPERS***));)

{

**while** (**true**)

{

StudentPaper studentPaper = (StudentPaper)(infile.readObject());

*gradeWithAllRubrics*(studentPaper);

studentFile.writeObject(studentPaper);

}

}

}

}

**catch** (EOFException ex)

{

System.***out***.println("EOF reached in " + ***STANDARD\_UNGRADED\_STUDENT\_PAPERS***);

}

**catch** (FileNotFoundException ex)

{

System.***out***.println("FileNotFoundException");

ex.printStackTrace();

}

**catch** (IOException ex)

{

System.***out***.println("IOException");

ex.printStackTrace();

}

**catch** (ClassNotFoundException ex)

{

System.***out***.println("ClassNotFoundException");

ex.printStackTrace();

}

**finally**

{

System.***out***.println

("Grading complete; records stored as StudentPaper objects in " +

***STANDARD\_GRADED\_STUDENT\_PAPERS***);

}

}

### 4.5.3 Code Showing *stream*() and Lambdas (separate int0 4.5.3 and 4.5.4 if you wish)

.

There are a few places where this is done. The following is from *StudentPaper.compareAnswerWithBounds* ().

**public** **void** compareAnswerWithBounds (**int** aMinimum, **int** aMaximum) {

/\* Postcondition: Input (student answers) were screened for aMinimum

\* and aMaximum; only numbers between these were accepted. The latter

\* were reported to console without breaks.

\*/

System.***out***.println("Entries within bounds "

+ "(an example...modify in StudentPaper; not separated): ");

studentAnswer

.stream()

.filter(w -> (w.intValue() >= aMinimum))

.filter(w -> (w.intValue() <= aMaximum))

.sorted()

.forEach(System.***out***::print);

System.***out***.print("\n");

}

## 4.7 YOUR CODE

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

<Your response here>

## 4.8 Evaluation

