Object relationship And File IO

Write a program to perform statistical analysis of scores for a class of students.The class may have up to 40 students.There are five quizzes during the term. Each student is identified by a four-digit student ID number.

The program is to print the student scores and calculate and print the statistics for each quiz. The output is in the same order as the input; no sorting is needed. The input is to be read from a text file. The output from the program should be similar to the following:

Here is some sample data (not to be used) for calculations:

Stud	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
1234	78	83	87	91	86
2134	67	77	84	82	79
1852	77	89	93	87	71
High Score	78	89	93	91	86
Low Score	67	77	84	82	71
Average	73.4	83.0	88.2	86.6	78.6

The program should print the lowest and highest scores for each quiz.

Plan Of Attack

Learning Objectives

You will apply the following topics in this assignment:

- File Input/Output operations.
- Working and populating an array of objects.
- Wrapper Classes.
- Object Oriented Design and Programming.

Understanding requirements

Here is a copy of actual data to be used for input.

```
Stud Qu1 Qu2 Qu3 Qu4 Qu5
1234 052 007 100 078 034
2134 090 036 090 077 030
3124 100 045 020 090 070
4532 011 017 081 032 077
5678 020 012 045 078 034
6134 034 080 055 078 045
7874 060 100 056 078 078
8026 070 010 066 078 056
9893 034 009 077 078 020
1947 045 040 088 078 055
2877 055 050 099 078 080
3189 022 070 100 078 077
4602 089 050 091 078 060
5405 011 011 000 078 010
6999 000 098 089 078 020
```

Essentially, you have to do the following:

- Read Student data from a text file.
- Compute High, Low and Average for each quiz.
- Print the Student data and display statistical information like High/Low/Average..

Also, add a class, which implements a custom exception handler to handle the following condition:

If input file has more more than 40 records, then read first 40 records for calculating statistics and throw an exception, when reading the 41st record. No exception is raised, if input file has 40 or less records of student data.

Design

This program can be written in one class. But dividing the code into simple and modular classes based on functionality, is at the heart of Object Oriented Design.

You must learn the concepts covered in the class and find a way to apply. I want to have a clear understanding through your application that you understand why Abstract classes and Interfaces exist. For this part, instructor or TA will not provide any guidance:

- Designing Class Exception Management. You need to add appropriate classes and methods to handle requirements for this assignment.
- Finding opportunities to use Abstract Classes and Interfaces.

Please make sure that you put each class in its own .java file.

```
package lab2;
class Student {
  private int SID;
  private int scores[] = new int[5];
  //write public get and set methods for
  //SID and scores
  //add methods to print values of instance variables.
}
/*********************************************************************************/
package lab2;
class Statistics
{
        int [] lowscores = new int [5];
        int [] highscores = new int [5];
        float [] avgscores = new float [5];
        void findlow(Student [] a){
           /*This method will find the lowest score and store it in
           an   array names lowscores. */
        }
         void findhigh(Student [] a){
           /* This method will find the highest score and store it in
           an     array names highscores. */
        }
        void findavg(Student [] a){
```

```
            /* This method will find avg score for each quiz and store
            it in an array names avgscores. */

        }
        //add methods to print values of instance variables.
}
*********************************************************************************/
package lab2;
class Util {
      static Student [] readFile(String filename, Student [] stu) {
      //Reads the file and builds student array.
      //Open the file using FileReader Object.
      //In a loop read a line using readLine method.
      //Tokenize each line using StringTokenizer Object
      //Each token is converted from String to Integer using parseInt
      method
      //Value is then saved in the right property of Student Object.
      }
}

*********************************************************************************/
//Putting it together in driver class:
       public static void main(String [] args) {
            Student lab2 [] = new Student[40];
            //Populate the student array
            lab2 = Util.readFile("filename.txt", lab2)
            Statistics statlab2 = new Statistics();
            statlab2.findlow(lab2);
            //add calls to findhigh and find average
            //Print the data and statistics
       }
```

Topics to Learn

Working with Text files

```
//ReadSource.java -- shows how to work with readLine and FileReader
public class ReadSource {
    public static void main(String[] arguments) {
      try {
            FileReader file = new
      FileReader("ReadSource.java");
```

```
            BufferedReader buff = new
                 BufferedReader(file);
             boolean eof = false;
             while (!eof) {
                 String line = buff.readLine();
                 if (line == null)
                    eof = true;
                 else
                     System.out.println(line);
             }
             buff.close();
         } catch (IOException e) {
             System.out.println("Error -- " + e.toString());
         }
    }
}

//How do you tokenize a String? You can use other ways of doing this,
if you like.
     StringTokenizer st = new StringTokenizer("this is a test");
     while (st.hasMoreTokens()) {
         System.out.println(st.nextToken());
     }

//How to convert a String to an Integer
     int x = Integer.parseInt(String) ;
```

Submitting your work

Before submitting your work, please reflect on following points:

1. No errors, program always works correctly and meets the specification(s) (2 points).
2. The code could be reused as a whole or each routine could be reused (2 points).
3. Concepts of OOD are applied (3 points):
    a. Usage of OOP constructs, relationships, interface and abstract classes.
    b. Code organized into functionally relevant packages - e.g. Exception, Util, Model etc. in their own respective packages.
    c. File IO API applied.
4. Advanced OOD constructs are applied/used (4 points):
    a. Usage of Interfaces
    b. Usage of Abstract Classes
    c. Custom Exception Handler.
5. Need to follow Java coding conventions (2 points).
6. Code Readability (as suggested in class) (2 points).
7. Adequately tested (unique test cases, covering boundary conditions) (2 points).
8. Class diagram is provided (UML usage is not required) (3 points).