Even Semester (2023)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information:

Surname: Djunaedy **Given Name:** Vania Agnes **Student ID:** 2602158531

Course Code : COMP6699001 **Course Name :** Object Oriented Programming

Class : L2BC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignment : Final Project Report

Submission Pattern

Due Date **:** 16 June 2023 **Submission Date :** 16 June 2023

The assignment should meet the below requirements.

I. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
II. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
III. The above information is complete and legible.
IV. Compiled pages are firmly stapled.
V. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

Vania A. D

Table of Contents

I. Introduction

A. Background

Back in our Object Oriented Programming class, we were assigned the challenge of writing a Java program that goes above and beyond what was taught in class. With that in mind, I had a difficult time deciding what to make for my final project. I finally opted to construct a "Student Management System" after considering several different topics for my final project. This system makes use of a lot of functionality that I never fully studied in class and is by far the most complex program I've created so far.

B. Problem Identification

The use of effective software solutions in student management can considerably improve and easen the process of administrative operations and communication between educational institutions, administrators, and students. The goal of this “Student Management System” is to create a program that optimizes the management and the maintenance of a student's data such as their personal information like: Full Name, Gender, Address, Parent’s Name, Phone Number and Birth Date and even their academic records in several subjects like: Science, Social, Math, Religion, Linguistic, and Ethics to boost their overall experience for both students and administrators.

There are frequently issues connected with time inefficiencies and potential inaccuracies in manual student management methods. Student’s data and Student’s score are very prone to mistakes and error if done the manual way. These processes can be easen for both school administrative workers and students if used to reduce the chance of errors in data and to reduce the time taken to store the data.

Furthermore, the use of a Student Management System removes the need for unnecessary paperwork as well as the additional expenses needed with manual record recording. The Program can store and save the data accurately, ensuring the accuracy of student records.

II. Project Specification

A. Program Name

The program I created goes by the name "Student Management System," which appropriately describes its objective of effectively managing student data. I chose this name deliberately to underline the program's utility and simplicity of use for users, particularly those who are using it for the first time. This system's principal goal is to provide effective management of student information while ensuring simplicity and clarity in its use. By using a simple name, I hope to create a user-friendly experience that allows for easy navigation and improves overall usage.

B. Description of the Program

The Student Management System is intended to make administrative work easier for school personnel and students by offering an effective platform for entering personal information and managing academic records. Users may easily store their information with this program while limiting the possibility of errors.

The system includes nine features, such as:

1. Input New Student: Users can safely enter and preserve student information.
2. View Student List: A complete list of all enrolled students is available.
3. View Student Details: By entering student ID, users can view that specific student's data.
4. Update Existing Students: Users can make changes to existing student records if needed.
5. Remove Students: There is an option to remove students from the system.
6. Input Student Scores: Enter an existing student's scores
7. Remove Student Scores: Users can erase individual student scores.
8. View Student Scores: Once you input Student Scores, it can be viewed using this feature.
9. Quit: Exiting the program.

This system's implementation uses a number of OOP concepts, including abstract classes, interfaces, and polymorphism. Thus makes use of numerous classes, such as ScoreRecordDao, StudentDao, Model, ScoreRecord, Student, Table, and the Main class. It also includes two .txt files: "grade-record.txt" for storing existing student scores and "student.txt" for storing student data based on input.

C. Program Libraries/ Modules

- java.text.SimpleDateFormat : This is used to parse dates according to a specific pattern, and this provides a convenient way to work with dates in general.
- java.util.Calendar : This provides a set of methods for working with dates, such as subtracting time units and retrieving specific component of date (day, month, year, etc)

- java.util.Scanner : This is used to provide reading input from various sources such as the consoles. By implementing this it makes it easier to handle user input and process data from external sources.
- java.util.ArrayList :This is part of the Java Collection Framework and used to create/implement ArrayList in your code.
- java.io.BufferedReader : It provides a method to read characters, lines, or entire files by buffering the input.
- java.io.BufferedWriter : This is used to write text to a character output stream, and it provides a method to write characters, lines etc.
- java.io.File : This represents a file or directory path in the file system, this is used for the .txt file we have in our program.
- java.io.FileReader : This is used to read characters from a file, it provides a method to read characters individually.
- java.io.FileWriter : This is used to write characters to a file.
- java.io.IOException : This is an exception class that is used when the I/O operation fails or is interrupted.
- java.util.UUID : This is used to make UUID, to make sure that the data is individually unique we generate random UUID to each student's data.
- java.util.Date : This is used to represent a specific point in time, to manipulate and format dates.

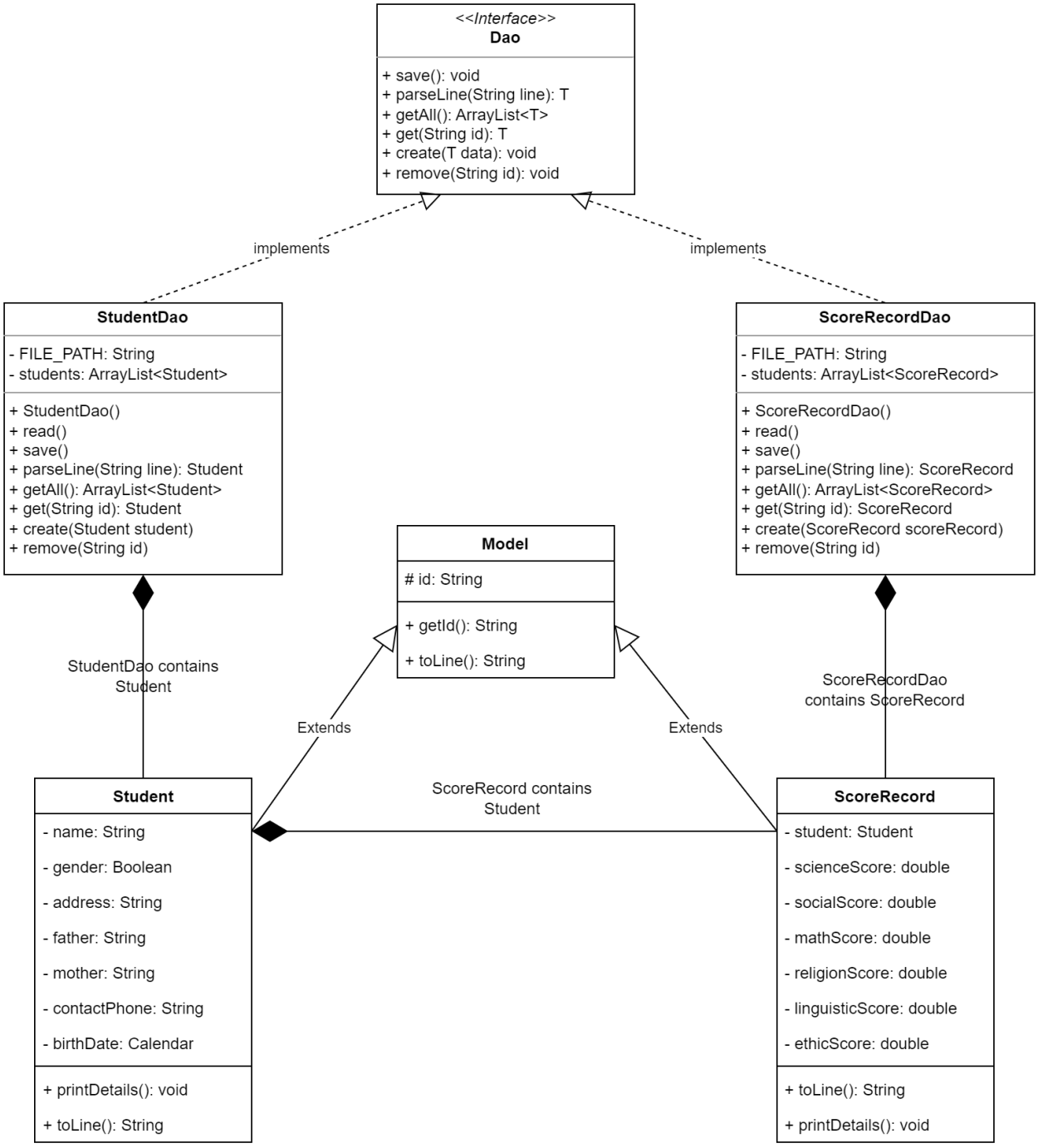
D. Program Files

- grade-record.txt : This file contains the data entered by the users for the student score data.
- students.txt : This file contains the data entered by the users for the personal student's data.
- Dao.java : The Dao interface is declaring the methods for the basic function of this system.
- ScoreRecordDao.java : The ScoreRecordDao class implements the Dao interface, so every method in Dao is implemented in this class. This class handles the function in regard for storing the student's scores.
- StudentDao.java : The StudentDao class implements the Dao interface, so every method in Dao is implemented in this class. This class handles the function in regard to storing student's personal data.

- Model.java : This is an abstract class that consists of attributes and methods that a model should have.
- ScoreRecord.java : This ScoreRecord class extends the Model Abstract class, meaning the methods are now inherited. This class is responsible for storing and managing the academic scores of a student.
- Student.java : This Student class extends the Model Abstract class, meaning the methods are now inherited. This class is responsible for storing and managing the personal data of a student.
- Table.java : this Table class functionality is to print the formatted tables in the program for features 2 and 8 in the program.
- Main.Java : This is the part where everything is runned from, basically the main file.

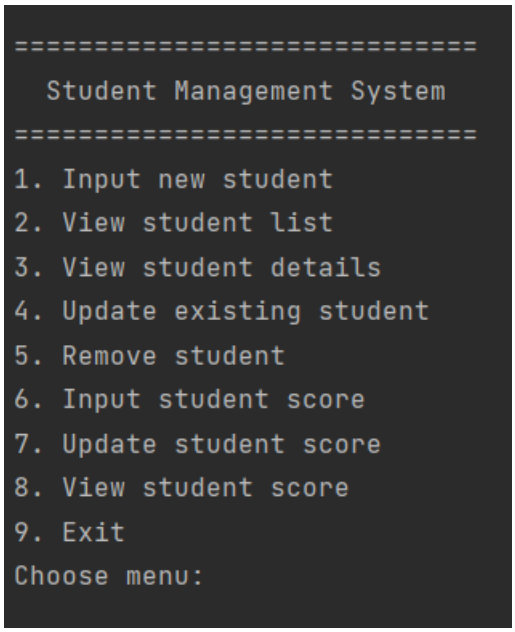
III. Solution Design

A. UML Class Diagram

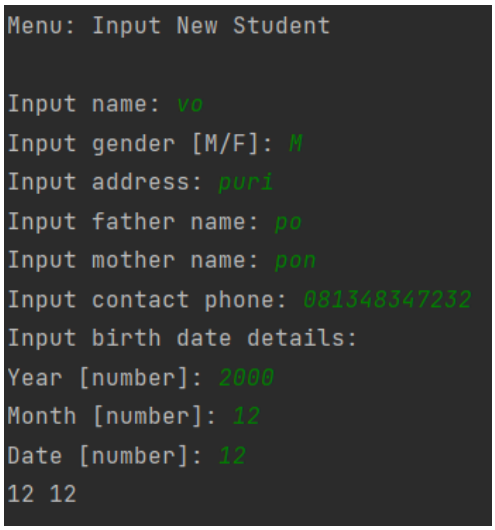


From here we can see that the Student Class and Score Record Class extends the Model Abstract Class. While StudentDao and ScoreRecordDao implement the Dao Interface thus using all the methods that are in the Dao Interface.

B. Screenshot of the Program



This is the "Student Management System" home page; the (CLI) indicates that it will be executed from the command line. between here, you can select between nine features in the intention of easing the administrative process.



This is the program's first feature, "Input New Student," and you can access it by typing 1. Then enter the student's personal information, such as name, gender, address, father's name, mother's name, contact phone number, and birth date. You have successfully entered the student's details.

```
Menu: View Students

STUDENT TABLE
+----------------+---------------------------+--------+--------------+----------------------+----------------------+-----------------+
| ID             | Full name                 | Gender | Birth Date   | Father               | Mother               | Contact Phone   |
+----------------+---------------------------+--------+--------------+----------------------+----------------------+-----------------+
| 1686858632992  | chelo                     | Female | 02-02-1999   | mike                 | vo                   | 081738578291    |
+----------------+---------------------------+--------+--------------+----------------------+----------------------+-----------------+
| 1686860964155  | dor                       | Male   | 23-08-1999   | bob                  | mary                 | 08184938498     |
+----------------+---------------------------+--------+--------------+----------------------+----------------------+-----------------+
| 1686927721517  | vo                        | Male   | 12-12-2000   | po                   | pon                  | 081348347232    |
+----------------+---------------------------+--------+--------------+----------------------+----------------------+-----------------+
```

You can access the program's second feature, "View Students List," by typing 2. The application will then display a table containing all of the data that the users have entered. As a unique identifier, this data assigns a random ID to all submitted data.

```
Menu: View Student Details

Input student id: 1686927721517

Student details:
ID              : 1686927721517
Name            : vo
Birth date      : 12-12-2000
Gender          : Male
Father name     : po
Mother name     : pon
Contact phone   : 081348347232
Address         : puri
```

This is the third feature of the program, "View Students Details," and you may access it by typing 3. What distinguishes this from the second feature is that you can only view one student's personal info by entering their ID.

```
Menu: Update Existing Student

Input student id: 1686860964155

Current student details:
ID              : 1686860964155
Name            : dor
Birth date      : 23-08-1999
Gender          : Male
Father name     : bob
Mother name     : mary
Contact phone   : 08184938498
Address         : pik

New student details:
Input name [Press ENTER to skip]:
Input gender [M/F] [Press ENTER to skip]:
Input address [Press ENTER to skip]:
Input father name [Press ENTER to skip]: ba
Input mother name [Press ENTER to skip]:
Input contact phone [Press ENTER to skip]:
Input birth date details:
Year [number] [Input -1 to skip]: -1
Month [number] [Input -1 to skip]: -1
Date [number] [Input -1 to skip]: -1
```

"Update Existing Student," the program's fourth feature. This feature is now used to update the program's already existing data. Users can use this feature after inputting 4. The existing student details will be shown to the users to help them understand what data to alter. To alter the data, simply write in the data and hit enter to skip. However, to skip the birth date, type -1 instead.

```
Menu: Remove Student

Input student id: 1686927721517

Student records has been remove successfully
```

This is the fifth feature of the program, and to access it, type 5. This feature is now available to assist users with removing student data from the program. Users can delete a student's data by entering their ID.

```
Choose menu: 6

Menu: Input Student Score

Input student id: 1686860964155

Input science score: 70
Input social score: 80
Input math score: 50
Input linguistic score: 60
Input religion score: 70
Input ethic score: 60
Student's score has been recoreded successfully
```

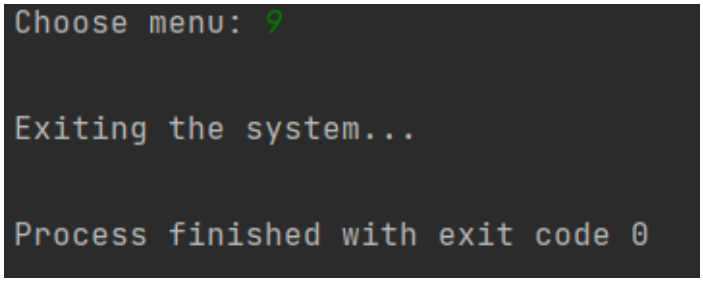
"Input Student Score" is the sixth feature of the program, and you can access it by typing 6. Users will be requested to fill up the score for each topic after entering an existing student's ID. You will be alerted that "Student's score has been recorded successfully" after successfully filling out each score.

```
Menu: Update Student Score

Input student id: 1686860964155

Current score score:
Student id          : 1686860964155
Student name        : dor
Science score       : 70.0
Social score        : 80.0
Math score          : 50.0
Linguistic score    : 60.0
Religion scpre      : 70.0
Ethic score         : 60.0

Input science score [Input -1 to skip]: -1
Input social score [Input -1 to skip]: -1
Input math score [Input -1 to skip]: -1
Input linguistic score [Input -1 to skip]: -1
Input religion score [Input -1 to skip]: 100
Input ethic score [Input -1 to skip]: -1
```

"Update Student Score," the program's seventh feature, is accessed by typing 7. The user will be requested to provide an existing student's ID, after which the user will be shown the current score for all subjects of that student to make updating the score easier. To modify the score, simply input the desired number, and to skip, simply type "-1".

```
Choose menu: 8

STUDENT SCORE TABLE
+----------------------------+-------------+-------------+-------------+-------------+-------------+-------------+
| Student name               | Science     | Social      | Math        | Religion    | Linguistic  | Ethic       |
+----------------------------+-------------+-------------+-------------+-------------+-------------+-------------+
| chelo                      | 70.0        | 80.0        | 90.0        | 600.0       | 50.0        | 40.0        |
+----------------------------+-------------+-------------+-------------+-------------+-------------+-------------+
| dor                        | 70.0        | 80.0        | 50.0        | 70.0        | 60.0        | 60.0        |
+----------------------------+-------------+-------------+-------------+-------------+-------------+-------------+
```

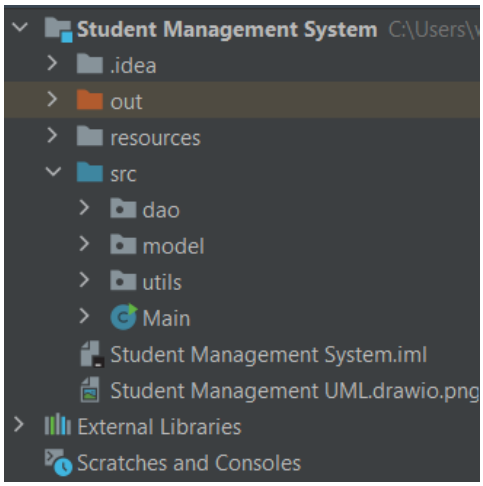
"View Student Score" is the program's eighth feature. You can gain access to this by typing 8. Users will be provided a table containing all of the previously inputted scores. It will also display the student's name.



Finally, there is the "Exit" option in this program; if you are content with all of the other features and wish to exit, simply type 9. And you will be kicked out of the system.

IV. Code Design and Explanation

A. Project Structure



In this project, all the .txt files are located at the resources folder.

While in the src folder, it consists of three packages: dao, model and utils. For the dao package it has the Dao Interface, "Studentdao" class and "ScoreRecordDao" class.

For the model package it has the "Model" abstract class, "ScoreRecord" class and "Student" class lastly in the utils package it has the "Table" class. And last but not least is the "Main"

B. Code Explanation (Dao Package)

- Dao Interface

```java
package dao;

import java.util.ArrayList;

/**
 * @interface Dao
 *
 * Consist of the method that an Object's Dao should implement.
 */
2 usages  2 implementations
public interface Dao<T> {
    2 usages  2 implementations
    public void read();
    5 usages  2 implementations
    public void save();
    2 usages  2 implementations
    public T parseLine(String line);
    2 usages  2 implementations
    public ArrayList<T> getAll();
    8 usages  2 implementations
    public T get(String id);
    2 usages  2 implementations
    public void create(T data);
    2 usages  2 implementations
    public void remove(String id);
}
```

Now, in the first package called "Dao," there is an interface called "Dao" (Data Access Object) with methods like read(), save(), parseLine(String line), getAll(), get(String id), create(T data), and remove(String id). This function will be added to the class that implements the Dao interface later. Because the T> is a generic data parameter, the interface can be used for a variety of objects.

- “ScoreRecordDao” Class

```
package dao;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import model.ScoreRecord;
// score record implements the dao interface
public class ScoreRecordDao implements Dao<ScoreRecord> {
// holds the file path
 private String FILE_PATH = "resources/grade-record.txt";
 private ArrayList<ScoreRecord> gradeRecords;
 private StudentDao studentDao;
```

Now in this section shows the package dao; meaning it’s from the dao package. Next is all the imports that are implemented in this code from java.io and java.util. From the word “implements” in the code it can be seen that this code implements the Dao interface, meaning all of the methods that were in the interface should be used. The FILE_PATH variables store the path to a text file (grade-record.txt) where the grade records inputted is stored. Variable studentDao ensures that if the code requires, they can retrieve data from that class. And lastly an ArrayList used to store the Score Record.

```
public ScoreRecordDao() {
 this.gradeRecords = new ArrayList<ScoreRecord>();
 this.studentDao = new StudentDao();

 File file = new File(FILE_PATH);

 if (file.exists()) {
   System.out.println("grade-record.txt exist!");
   this.read();
 } else {
```

```
    System.out.println("grade-record.txt created!");
    try {
      file.createNewFile();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

This method is used to create a new file using the FILE_PATH variables. It also checks where the file exists or not. If it exists then it prints a message, before calling the read() method to read the grade records from the file. If it doesn't exist then it would create a new file.

```
@Override
public void save() {
 try {
        BufferedWriter writer = new BufferedWriter(new
FileWriter(FILE_PATH));

   StringBuilder builder = new StringBuilder();

   this.gradeRecords.forEach(gradeRecord -> {
     builder.append(gradeRecord.toLine()).append("\n");
   });

   writer.write(builder.toString());

   writer.close();
 } catch (IOException e) {
   e.printStackTrace();
 }
}
```

The ScoreRecordDao class's save() method is in charge of storing the grade records stored in the gradeRecords list back to the file given by FILE_PATH.

```
@Override
public ScoreRecord parseLine(String line) {
 String tokens[] = line.split(";");

 String id = tokens[0];
 String studentId = tokens[1];
 double scienceScore = Double.parseDouble(tokens[2]);
 double socialScore = Double.parseDouble(tokens[3]);
 double mathScore = Double.parseDouble(tokens[4]);
```

```
 double religionScore = Double.parseDouble(tokens[5]);
 double linguisticScore = Double.parseDouble(tokens[6]);
 double ethicScore = Double.parseDouble(tokens[7]);

  return new ScoreRecord(id, this.studentDao.get(studentId),
scienceScore, socialScore, mathScore,
     religionScore, linguisticScore, ethicScore);
}
```

The ScoreRecordDao class's parseLine() method is in charge of transforming a line of text representation into a ScoreRecord object.

```
 @Override
 public ArrayList<ScoreRecord> getAll() {
   return this.gradeRecords;
 }
// see an object based on it's id
 @Override
 public ScoreRecord get(String id) {
   for (int i = 0; i < this.gradeRecords.size(); i++) {
     if (this.gradeRecords.get(i).getId().equals(id)) {
       return this.gradeRecords.get(i);
     }
   }

   return null; // if the grade doesn't exist!
 }
```

The ScoreRecordDao class's getAll() method simply returns the gradeRecords list, which contains all ScoreRecord objects. It allows rh to obtain all of the records contained in the DAO. The get(String id) returns a specific ScoreRecord object depending on its id.

```
public ScoreRecord getByStudentId(String studentId) {
 for (int i = 0; i < this.gradeRecords.size(); i++) {
                                                        if
(this.gradeRecords.get(i).getStudent().getId().equals(student
Id)) {
     return this.gradeRecords.get(i);
   }
 }

 return null;
}
```

This is used to obtain data based on the Id. It is utilized in many features that require the Id to be entered first.

```
@Override
public void create(ScoreRecord gradeRecord) {
 this.gradeRecords.add(gradeRecord);
 this.save();
}
```

The "create" method, as the name implies, is used to update an inputted score. After it has added the new data it will save it.

```
@Override
 public void remove(String id) {
   ScoreRecord gradeRecord = this.get(id);
   if (gradeRecord != null) {
     this.gradeRecords.remove(gradeRecord);
     this.save();
   }
 }
}
```

The function for this one is quite simple, like the name suggest it will remove an object based on its Id.

- “StudentDao” Class

```
package dao;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;

import model.Student;

// Student Data Access Object
public class StudentDao implements Dao<Student> {
 private String FILE_PATH = "resources/students.txt";
 // stores the data of the student
 private ArrayList<Student> students;
```

From here you could see that this is also from the package “dao”. It also used imports from java.io and java.util. From this, we can see that this class also implements the Dao Interface. The FILE_PATH variables store

the path to a text file (student.txt) containing the student's personal data. This class likewise uses an ArrayList to hold the student's data.

```
public StudentDao() {
 this.students = new ArrayList<Student>();

 File file = new File(FILE_PATH);

 if (file.exists()) {
   // Load the file if file exist.
   System.out.println("students.txt exist!");
   this.read();
 } else {
   // Create the file if file doesn't exist
   System.out.println("students.txt created!");
   try {
     file.createNewFile();
   } catch (IOException e) {
     e.printStackTrace();
   }
 }
}
```

This is the same as before in the “ScoreRecordDao” class but this is the student’s version.

```
@Override
 public void read() {
   StringBuilder fileContent = new StringBuilder();
   try {
     // Reading the file (load all of the data from the file)
          BufferedReader reader = new BufferedReader(new
FileReader(FILE_PATH));

     String line;
     while ((line = reader.readLine()) != null) {
       fileContent.append(line).append("\n");
     }

     // If the file is not empty
     if (fileContent.toString().length() != 0) {
       String lines[] = fileContent.toString().split("\n");
       for (int i = 0; i < lines.length; i++) {
           // parse the line to Student object, then add the
object to `students`.
         students.add(this.parseLine(lines[i]));
       }
     }
```

```
    reader.close();
   } catch (IOException e) {
    e.printStackTrace();
   }
 }

 // Save the students ArrayList data to the file
 @Override
 public void save() {
   try {
      // Writing the file (save all of the data at students
ArrayList to the file)
          BufferedWriter writer = new BufferedWriter(new
FileWriter(FILE_PATH));

    StringBuilder builder = new StringBuilder();

    this.students.forEach(student -> {
      builder.append(student.toLine()).append("\n");
    });

    writer.write(builder.toString());

    writer.close();
   } catch (IOException e) {
    e.printStackTrace();
   }
 }

 // parse a single line of student data stored in file to
Student object
 @Override
 public Student parseLine(String line) {
   String tokens[] = line.split(";");

   String id = tokens[0];
   String name = tokens[1];
   boolean gender = Integer.valueOf(tokens[2]) == 1 ? true :
false;
   String address = tokens[3];
   String father = tokens[4];
   String mother = tokens[5];
   String contactPhone = tokens[6];
   Calendar birthDate = Calendar.getInstance();
                    birthDate.set(Integer.valueOf(tokens[9]),
Integer.valueOf(tokens[8]), Integer.valueOf(tokens[7]));
```

```
    return new Student(id, name, gender, address, father,
mother, contactPhone, birthDate);
  }

  // Retrieve all of the students data
  @Override
  public ArrayList<Student> getAll() {
    return this.students;
  }

  // Retrieve a single student data based on it's id
  @Override
  public Student get(String id) {
    for (int i = 0; i < this.students.size(); i++) {
      if (this.students.get(i).getId().equals(id)) {
        return this.students.get(i);
      }
    }

    return null; // if the studentId doesn't exist!
  }

  // Store a new Student object into the students list
  @Override
  public void create(Student student) {
    this.students.add(student);
    this.save();
  }

  // Remove a student based on the given student id
  @Override
  public void remove(String id) {
    Student student = this.get(id);
    if (student != null) {
      this.students.remove(student);
      this.save();
    }
  }
}
```

This code does the same job as the "ScoreRecordDao" class, but it is designed for the feature that is for the student's data rather than the student's score.

C. Code Explanation (Model Package)

- "Model" Abstract Class

```java
package model;

/**
* @abstract Model
*
* An abstract class Model that consist of attribute and
method
* that an Object's Model should have. The standard is to
have
* id attributes and getId() method.
*/
public abstract class Model {
 protected String id;

 public String getId() {
   return id;
 }

 public abstract String toLine();
}
```

So this is the abstract class that is implemented in the program; it is called "Model," and its purpose is to serve as a foundation for all other classes.

- "Score Record" Class

```java
package model;

import java.util.UUID;

public class ScoreRecord extends Model {
 private Student student;
 private double scienceScore, socialScore, mathScore,
religionScore, linguisticScore, ethicScore;
```

We can see from here that it comes from the "model" package. It also includes java.util.UUID that is used to produce random UUIDs in order to provide a unique classifier for any data submitted into this software. It also has the variable student, so we can access student data if necessary. Finally, it initialized all of the variables in this class.

```
public ScoreRecord() {
 this.id = UUID.randomUUID().toString();
 this.student = null;
 this.scienceScore = this.socialScore = this.mathScore =
this.religionScore = this.linguisticScore = this.ethicScore =
0;
}

//takes the student object as parameters and initialize the
instance variables
public ScoreRecord(Student student, double scienceScore,
double socialScore, double mathScore, double religionScore,
   double linguisticScore, double ethicScore) {
 this.id = UUID.randomUUID().toString();
 this.student = student;
 this.scienceScore = scienceScore;
 this.socialScore = socialScore;
 this.mathScore = mathScore;
 this.religionScore = religionScore;
 this.linguisticScore = linguisticScore;
 this.ethicScore = ethicScore;
}

public ScoreRecord(String id, Student student, double
scienceScore, double socialScore, double mathScore,
   double religionScore, double linguisticScore, double
ethicScore) {
 this.id = id;
 this.student = student;
 this.scienceScore = scienceScore;
 this.socialScore = socialScore;
 this.mathScore = mathScore;
 this.religionScore = religionScore;
 this.linguisticScore = linguisticScore;
 this.ethicScore = ethicScore;
}
```

Now this code right here shows you polymorphism, starting with the first method it is used as a default constructor and by any chance no input was given. Second constructor is used to take the student object as parameters and initialize the instance variables. The last constructor fills the object's attributes with the values you supply, allowing you to store and access the score records for a specific student.

```
@Override
public String toLine() {
 String line = String.format("%s;%s;%f;%f;%f;%f;%f;%f",
     id,
     student.getId(),
     scienceScore,
     socialScore,
     mathScore,
     religionScore,
     linguisticScore,
     ethicScore);
 return line;
}
```

This is the method that was available in the abstract class “Model” that is used to create a String line following that specific format.

```
public Student getStudent() {
 return student;
}

public void setStudent(Student student) {
 this.student = student;
}

public double getScienceScore() {
 return scienceScore;
}

public void setScienceScore(double scienceScore) {
 this.scienceScore = scienceScore;
}

public double getSocialScore() {
 return socialScore;
}

public void setSocialScore(double socialScore) {
 this.socialScore = socialScore;
}

public double getMathScore() {
 return mathScore;
}

public void setMathScore(double mathScore) {
 this.mathScore = mathScore;
}
```

```
public double getReligionScore() {
 return religionScore;
}

public void setReligionScore(double religionScore) {
 this.religionScore = religionScore;
}

public double getLinguisticScore() {
 return linguisticScore;
}

public void setLinguisticScore(double linguisticScore) {
 this.linguisticScore = linguisticScore;
}

public double getEthicScore() {
 return ethicScore;
}

public void setEthicScore(double ethicScore) {
 this.ethicScore = ethicScore;
}
```

This is the getter setter of this class

```
public void printDetails() {
            System.out.printf("Student    id\t\t\t:    %s\n",
this.student.getId());
            System.out.printf("Student    name\t\t:    %s\n",
this.student.getName());
           System.out.printf("Science    score\t\t:    %s\n",
this.scienceScore);
            System.out.printf("Social    score\t\t:    %s\n",
this.socialScore);
            System.out.printf("Math    score\t\t\t:    %s\n",
this.mathScore);
          System.out.printf("Linguistic    score\t:    %s\n",
this.linguisticScore);
          System.out.printf("Religion    scpre\t\t:    %s\n",
this.religionScore);
            System.out.printf("Ethic    score\t\t\t:    %s\n",
this.ethicScore);
 }
}
```

This is a method that is used to print the details used when asking about the scores.

- "Student" Class

```
package model;
import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class Student extends Model {
 private String name;
 private boolean gender; // 1 -> male, 0 -> female
 private String address;
 private String father;
 private String mother;
 private String contactPhone;
 private Calendar birthDate;
```

From here it shows that this class also extends from the Abstract class called "Model" and also initializes some variables used for the input of student's data.

```
public Student() {
 this.id = String.valueOf(new Date().getTime());
 this.name = "";
 this.gender = true;
 this.address = this.father = this.mother = this.contactPhone
= "";
 this.birthDate = Calendar.getInstance();
}

public Student(String name, boolean gender, String address,
String father, String mother, String contactPhone, Calendar
birthDate) {
 this.id = String.valueOf(new Date().getTime());
 this.name = name;
 this.gender = gender;
 this.address = address;
 this.father = father;
 this.mother = mother;
 this.contactPhone = contactPhone;
 this.birthDate = birthDate;
}
public Student(String id, String name, boolean gender, String
address, String father, String mother, String contactPhone,
Calendar birthDate) {
```

```
 this.id = id;
 this.name = name;
 this.gender = gender;
 this.address = address;
 this.father = father;
 this.mother = mother;
 this.contactPhone = contactPhone;
 this.birthDate = birthDate;
}
```

This is also an example of polymorphism, the first constructor is used when the user's only want to initialize an empty student. The second constructor will be used when creating a new student's data. The last constructor will be used when we load Student data from the file

```
@Override
 public String toLine() {
                                String          line          =
String.format("%s;%s;%d;%s;%s;%s;%s;%d;%d;%d",
     id,
     name,
     gender ? 1 : 0,
     address,
     father,
     mother,
     contactPhone,
     birthDate.get(Calendar.DATE),
     birthDate.get(Calendar.MONTH),
     birthDate.get(Calendar.YEAR)
   );

   return line;
 }

 public String getName() {
   return name;
 }

 public void setName(String name) {
   this.name = name;
 }

 public boolean isGender() {
   return gender;
 }

 public void setGender(boolean gender) {
```

```
    this.gender = gender;
  }

  public String getAddress() {
    return address;
  }

  public void setAddress(String address) {
    this.address = address;
  }

  public String getFather() {
    return father;
  }

  public void setFather(String father) {
    this.father = father;
  }

  public String getMother() {
    return mother;
  }

  public void setMother(String mother) {
    this.mother = mother;
  }

  public String getContactPhone() {
    return contactPhone;
  }

  public void setContactPhone(String contactPhone) {
    this.contactPhone = contactPhone;
  }

  public Calendar getBirthDate() {
    return birthDate;
  }

  public void setBirthDate(Calendar birthDate) {
    this.birthDate = birthDate;
  }

  public void printDetails() {
    System.out.printf("ID\t\t\t\t: %s\n", this.id);
    System.out.printf("Name\t\t\t: %s\n", this.name);
    System.out.printf("Birth date\t\t: %s\n",
```

```
                                                                    new
SimpleDateFormat("dd-MM-yyyy").format((this.birthDate.getTime
()));
      System.out.printf("Gender\t\t\t:  %s\n",  this.gender  ?
"Male" : "Female");
   System.out.printf("Father name\t\t: %s\n", this.father);
   System.out.printf("Mother name\t\t: %s\n", this.mother);
            System.out.printf("Contact    phone\t:    %s\n",
this.contactPhone);
   System.out.printf("Address\t\t\t: %s\n", this.address);
 }
}
```

All of this code also shares the same resemblance and functionality to the "Score Record" Class but makes it Student version.

D. Code Explanation (Utils Package)

- "Table" Class

This code is used to create the table for feature 2 and 8

```
package utils;

public class Table {
 private int columnSize;
 private String columns[];
 private int size[];

 public Table(int columnSize, String columns[], int size[]) {
   this.columnSize = columnSize;
   this.columns = columns;
   this.size = size;
 }
 // print the separator of the table
 private void printSeparator() {
   for (int i = 0; i < this.columnSize; i++) {
     System.out.print("+");
     for (int j = 0; j < this.size[i]; j++) {
       System.out.printf("-");
     }
   }
   System.out.println("+");
 }
// print the header of the table
 public void printHeader() {
   this.printSeparator();
```

```
    System.out.print("| ");
    for (int i = 0; i < this.columnSize; i++) {
      System.out.printf(
          String.format("%-%ds | ", this.size[i] - 2),
          this.columns[i]);
    }
    System.out.println();
    this.printSeparator();
  }
// print the rows of the table
 public void printRows(String rows[]) {
    System.out.print("| ");
    for (int i = 0; i < this.columnSize; i++) {
      System.out.printf(
          String.format("%-%ds | ", this.size[i] - 2),
          rows[i]);
    }

    System.out.println();
    this.printSeparator();
  }
}
```

This code is used to create the table for feature 2 and 8. It consists of the header, row and separator which is all part of creating a table.

E. Main.Java

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Scanner;

import dao.ScoreRecordDao;
import dao.StudentDao;
import model.ScoreRecord;
import model.Student;
import utils.Table;

public class Main {
 private static Scanner scanner;
 private static StudentDao studentDao;
 private static ScoreRecordDao scoreRecordDao;
```

This shows all the imports used in the main code and some initialized variables.

```
public static void main(String[] args) {
 scanner = new Scanner(System.in);
```

```
studentDao = new StudentDao();
scoreRecordDao = new ScoreRecordDao();

while (true) {
  System.out.println();

  System.out.println("============================");
  System.out.println("  Student Management System");
  System.out.println("============================");
  System.out.println("1. Input new student");
  System.out.println("2. View student list");
  System.out.println("3. View student details");
  System.out.println("4. Update existing student");
  System.out.println("5. Remove student");
  System.out.println("6. Input student score");
  System.out.println("7. Update student score");
  System.out.println("8. View student score");
  System.out.println("9. Exit");
  System.out.print("Choose menu: ");

  int menu = scanner.nextInt();

  scanner.nextLine();

  System.out.println();
```

This is the printing process for the program's main menu, which you will see when you first launch it.

```
switch (menu) {
    case 1:
      menuInputNewStudent();
      break;
    case 2:
      menuViewStudents();
      break;
    case 3:
      menuViewStudentDetails();
      break;
    case 4:
      menuUpdateExistingStudent();
      break;
    case 5:
      menuRemoveStudent();
      break;
    case 6:
      menuInputStudentScore();
```

```
        break;
      case 7:
        menuUpdateStudentScore();
        break;
      case 8:
        menuViewStudentScore();
        break;
      case 9:
        System.out.println("Exiting the system...");
        return;
      default:
         System.out.println("The menu doesn't exist. Please input the
proper menu number!");
        break;
    }
  }
}
```

The switch statement is used to do various actions depending on the value of the menu variable. It evaluates the menu value and runs the associated block of code for the matched case. As you can see, there are 9 cases, all of which conform to the features. If users type any number from 1 to 9, they will be sent to that feature.

```
private static void menuInputNewStudent() {
 System.out.println("Menu: Input New Student\n");

 System.out.print("Input name: ");
 String name = scanner.nextLine();

 System.out.print("Input gender [M/F]: ");
 String gender = scanner.nextLine();

 System.out.print("Input address: ");
 String address = scanner.nextLine();

 System.out.print("Input father name: ");
 String father = scanner.nextLine();

 System.out.print("Input mother name: ");
 String mother = scanner.nextLine();

 System.out.print("Input contact phone: ");
 String contactPhone = scanner.nextLine();

 System.out.println("Input birth date details:");
 System.out.print("Year [number]: ");
 int year = scanner.nextInt();
```

```
 System.out.print("Month [number]: ");
 int month = scanner.nextInt();

 System.out.print("Date [number]: ");
 int date = scanner.nextInt();

 Calendar birthDate = Calendar.getInstance();

 System.out.printf("%d %d\n", month, date);
 birthDate.set(year, month-1, date);

 System.out.println();

 studentDao.create(
     new Student(
         name,
         gender.equals("M") ? true : false,
         address,
         father,
         mother,
         contactPhone,
         birthDate));
}
```

This is what is printed when feature 1 is chosen

```
private static void menuViewStudents() {
 System.out.println("Menu: View Students\n");

 System.out.println("STUDENT TABLE");
  String columns[] = { "ID", "Full name", "Gender", "Birth Date",
"Father", "Mother", "Contact Phone" };
 int size[] = { 18, 30, 8, 16, 24, 24, 18 };
 Table table = new Table(7, columns, size);

 table.printHeader();

 // get all of the students data and output it as a beautiful row
 studentDao.getAll().forEach(student -> {
   String row[] = {
       student.getId(),
       student.getName(),
       student.isGender() ? "Male" : "Female",
                                                                  new
SimpleDateFormat("dd-MM-yyyy").format((student.getBirthDate().getTi
me())),
       student.getFather(),
       student.getMother(),
       student.getContactPhone()
   };
```

```
    table.printRows(row);
  });
}
```

This is the code for the second feature, it will get all of the inputted student data and be displayed in a table.

```
private static void menuViewStudentDetails() {
  System.out.println("Menu: View Student Details\n");
  System.out.print("Input student id: ");
  String studentId = scanner.nextLine();

  System.out.println();

  Student student = studentDao.get(studentId);

  if (student == null) {
    System.out.println("Student id doesn't exist");
  } else {
    System.out.println("Student details:");
    student.printDetails();
  }
}
```

This will display the student info by inquiring about the student's Id. The data of the student will then be displayed. If the student's Id does not match anything in the system, the system will report that the student does not exist.

```
private static void menuUpdateExistingStudent() {
  System.out.println("Menu: Update Existing Student\n");
  System.out.print("Input student id: ");
  String studentId = scanner.nextLine();

  System.out.println();

  Student student = studentDao.get(studentId);

  if (student == null) {
    System.out.println("Student id doesn't exist");
  } else {
    System.out.println("Current student details:");
    student.printDetails();

    System.out.println("\nNew student details:");

    System.out.print("Input name [Press ENTER to skip]: ");
    String name = scanner.nextLine();
    if (name.length() != 0) {
      student.setName(name);
```

```
  }

  System.out.print("Input gender [M/F] [Press ENTER to skip]: ");
  String gender = scanner.nextLine();
  if (gender.length() != 0) {
    student.setGender(gender == "M" ? true : false);
  }

  System.out.print("Input address [Press ENTER to skip]: ");
  String address = scanner.nextLine();
  if (address.length() != 0) {
    student.setAddress(address);
  }

  System.out.print("Input father name [Press ENTER to skip]: ");
  String father = scanner.nextLine();
  if (father.length() != 0) {
    student.setFather(father);
  }

  System.out.print("Input mother name [Press ENTER to skip]: ");
  String mother = scanner.nextLine();
  if (mother.length() != 0) {
    student.setMother(mother);
  }

  System.out.print("Input contact phone [Press ENTER to skip]: ");
  String contactPhone = scanner.nextLine();
  if (contactPhone.length() != 0) {
    student.setContactPhone(contactPhone);
  }

  System.out.println("Input birth date details:");
  System.out.print("Year [number] [Input -1 to skip]: ");
  int year = scanner.nextInt();
  if (year != -1) {
    student.getBirthDate().set(Calendar.YEAR, year);
  }

  System.out.print("Month [number] [Input -1 to skip]: ");
  int month = scanner.nextInt();
  if (month != -1) {
    student.getBirthDate().set(Calendar.MONTH, month-1);
  }

  System.out.print("Date [number] [Input -1 to skip]: ");
  int date = scanner.nextInt();
  if (date != -1) {
    student.getBirthDate().set(Calendar.DATE, date);
```

```
    }

    studentDao.save(); // save updated data
  }
}
```

This is the fourth functionality, and it simply allows users to update existing data. It will display the current data information to assist the user in updating the data; to modify the data, simply input and to skip, simply hit enter. To skip the birth date column, press "-1" on your keyboard.

```
private static void menuRemoveStudent() {
 System.out.println("Menu: Remove Student\n");
 System.out.print("Input student id: ");
 String studentId = scanner.nextLine();

 System.out.println();

 Student student = studentDao.get(studentId);

 if (student == null) {
   System.out.println("Student id doesn't exist");
 } else {
   studentDao.remove(studentId);

                            ScoreRecord         scoreRecord         =
scoreRecordDao.getByStudentId(studentId);

   if (scoreRecord != null) {
     scoreRecordDao.remove(scoreRecord.getId());
   }

         System.out.println("Student   records   has   been   remove
successfully");
 }
}
```

As the name suggests this code is used to remove the student's data.

```
private static void menuInputStudentScore() {
 System.out.println("Menu: Input Student Score\n");
 System.out.print("Input student id: ");
 String studentId = scanner.nextLine();

 System.out.println();

 Student student = studentDao.get(studentId);
```

```
 if (student == null) {
   System.out.println("Student id doesn't exist");
   return;
 }

              ScoreRecord              scoreRecord              =
scoreRecordDao.getByStudentId(studentId);
 if (scoreRecord != null) {
       System.out.println("Student's  score  has  been  inputted
before.\nPlease do an update to the existing record.");
   return;
 }

 System.out.print("Input science score: ");
 double scienceScore = scanner.nextDouble();

 System.out.print("Input social score: ");
 double socialScore = scanner.nextDouble();

 System.out.print("Input math score: ");
 double mathScore = scanner.nextDouble();

 System.out.print("Input linguistic score: ");
 double linguisticScore = scanner.nextDouble();

 System.out.print("Input religion score: ");
 double religionScore = scanner.nextDouble();

 System.out.print("Input ethic score: ");
 double ethicScore = scanner.nextDouble();

 scoreRecordDao.create(
     new ScoreRecord(
         student,
         scienceScore,
         socialScore,
         mathScore,
         religionScore,
         linguisticScore,
         ethicScore));

    System.out.println("Student's   score   has   been   recorded
successfully");
}
```

This program's sixth feature allows you to update an existing student score. You will be prompted to enter the student's id before entering the student's score.

```
private static void menuUpdateStudentScore() {
 System.out.println("Menu: Update Student Score\n");
 System.out.print("Input student id: ");
 String studentId = scanner.nextLine();

 System.out.println();

 Student student = studentDao.get(studentId);

 if (student == null) {
   System.out.println("Student id doesn't exist");
   return;
 }

               ScoreRecord               scoreRecord               =
scoreRecordDao.getByStudentId(studentId);
 if (scoreRecord == null) {
   System.out
           .println("Student's score record doesn't exist. Can't be
updated.\nPlease input the student score first.");
   return;
 }

 System.out.println("Current score score:");
 scoreRecord.printDetails();

 System.out.println();

 System.out.print("Input science score [Input -1 to skip]: ");
 double scienceScore = scanner.nextDouble();
 if (scienceScore != -1) {
   scoreRecord.setScienceScore(scienceScore);
 }

 System.out.print("Input social score [Input -1 to skip]: ");
 double socialScore = scanner.nextDouble();
 if (socialScore != -1) {
   scoreRecord.setSocialScore(socialScore);
 }

 System.out.print("Input math score [Input -1 to skip]: ");
 double mathScore = scanner.nextDouble();
 if (mathScore != -1) {
   scoreRecord.setMathScore(mathScore);
 }

 System.out.print("Input linguistic score [Input -1 to skip]: ");
 double linguisticScore = scanner.nextDouble();
 if (linguisticScore != -1) {
```

```
    scoreRecord.setLinguisticScore(linguisticScore);
  }

  System.out.print("Input religion score [Input -1 to skip]: ");
  double religionScore = scanner.nextDouble();
  if (religionScore != -1) {
    scoreRecord.setReligionScore(religionScore);
  }

  System.out.print("Input ethic score [Input -1 to skip]: ");
  double ethicScore = scanner.nextDouble();

  if (ethicScore != -1) {
    scoreRecord.setEthicScore(ethicScore);
  }
}
```

This is quite similar to the menuUpdateExistingStudent but what makes them different is the fact that this is for scores.

```
private static void menuViewStudentScore() {
    System.out.println("STUDENT SCORE TABLE");
      String columns[] = { "Student name", "Science", "Social",
"Math", "Religion", "Linguistic", "Ethic" };
    int size[] = { 30, 14, 14, 14, 14, 14, 14 };
    Table table = new Table(7, columns, size);

    table.printHeader();

    scoreRecordDao.getAll().forEach(score -> {
      String row[] = {
          score.getStudent().getName(),
          String.valueOf(score.getScienceScore()),
          String.valueOf(score.getSocialScore()),
          String.valueOf(score.getMathScore()),
          String.valueOf(score.getReligionScore()),
          String.valueOf(score.getLinguisticScore()),
          String.valueOf(score.getEthicScore()),
      };

      table.printRows(row);
    });
  }
}
```

Lastly this is to view the score in a table.

V. Lesson Learned

This program has taught me numerous new libraries and modules that I have yet to investigate. I also learned how to create a command-line-based program known as CLI. This is the first time I've created a program this complex; overall, it's been a stressful but informative experience for me. This significantly advanced my knowledge of java coding.

VI. References

https://stackoverflow.com/
https://www.geeksforgeeks.org
https://www.youtube.com/

VII. Important Links

A. Video Link: https://drive.google.com/file/d/1BCMMA9oOfYHzyTKeNzZ5weFS-fp6BF5R/view?usp=sharing

B. Github Link: https://github.com/VaniaAgnes/OOP-Final-Project.git