

Object Oriented Programming

Final Project Report

“Student Track”



Matthew Staniswinata - 2702325962

LECTURER:

Jude Joseph Lamug Martinez MCS

School of Computing and Creative Arts

Computer Science Program

Bina Nusantara International University

Jakarta

2024

Table of Contents

Table of Contents.....	2
Background.....	3
Project Specification.....	4
Program Description.....	5
Class Diagram.....	5
Code Explanation and Implementation.....	6
Working Program.....	18
References.....	25

Background

One of the main problem occurring around the world is about education. Either it's good or bad thing for people to hear about the situation of education. People hear about education, and they will think about first the quality and inequality that's happening around the world and can be specific just to be around the nation.

The problem will start either by looking at students' and/or teachers' performances. Hereby continue on the progress of education, not all school have the great productivity and efficiency on managing students. Especially with schools that rely on a set of teachers that need to teach and manage students across the school, from elementary until high school.

In terms of efficiency and management, there are limitations looking through the traditional system such as paper-based which have many drawbacks and optimization limitation happening between students and teachers. That's why an innovative solution and ways to be open-minded for anyone to see the perspectives of the outside world and acquire the knowledge that would help to progress.

There are many limitations, such as repetitive and time-consuming nature where teachers must manually write down detailed information about each student, which can be a tedious task, especially for large schools with hundreds or thousands of students. Lack of efficiency, which the system relies solely on manual input by individual teachers, leading to inconsistencies and human errors in data management. Inconsistent grading, as in school usually different teachers may use unique scoring rubrics, resulting in disparities in student grades, which can lead to unfair judgments and misinterpretations.

Project Specification

This project is not the solution that will solve all of the limitations and drawbacks that happened to be the problem for the traditional management system.

This project is supposed to be the first step or insight for people to know and realize about what is this thing called student database management system software. As it will address the limitations, by having streamline data management, automating the process of any modifications towards the data from school and any student's data and would improve efficiency of progressing.

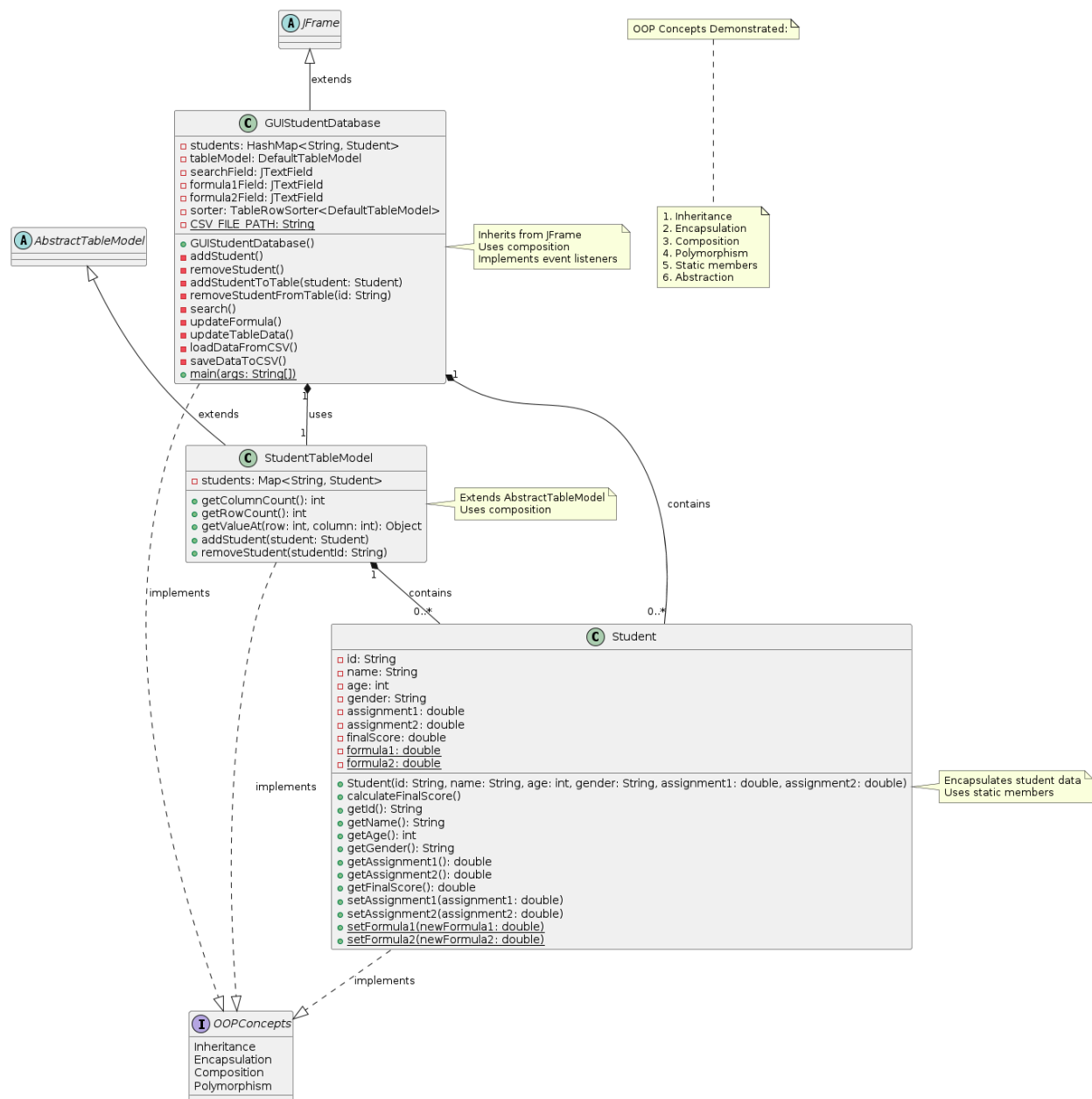
Enhancing standardization, with the idea of scores and weight of any assignments to be as the final score of the student. Having flexibility and real-time updates towards the data as they are saved into the file to be processed and managing every single student's information.

This project starts with implementation, of the "ideal" simple student database management system with basic functionality of sorting and searching between student's data and able to handle exception problems that might occur in using the software.

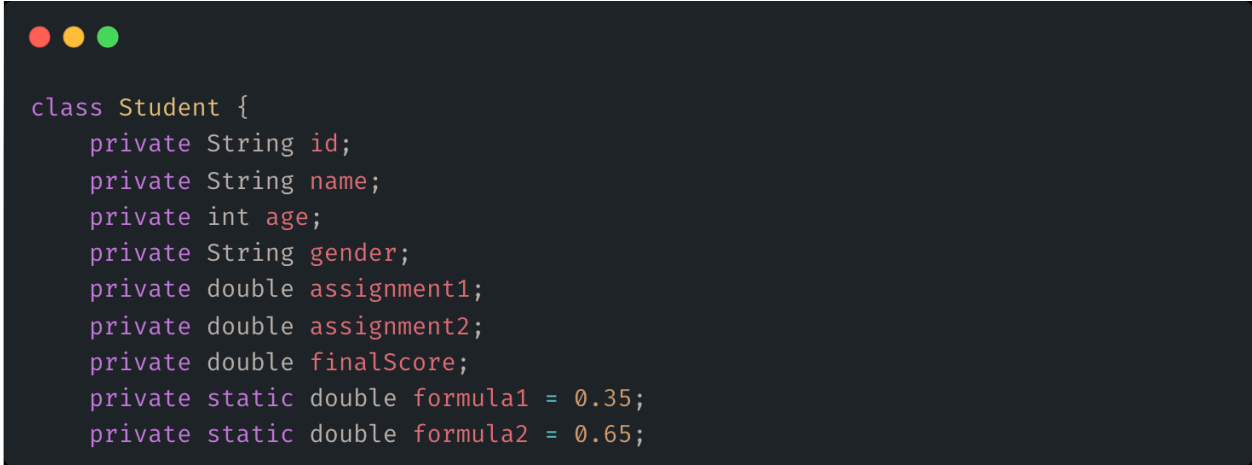
Program Description

“Student Track”, student database management system software that teachers would use to alternate or improve their progress. Everything is implemented using Java programming language, with the features and full scale GUI of the program to be immediately be able to be used by the user.

Class Diagram



Code Explanation and Implementation



```
class Student {  
    private String id;  
    private String name;  
    private int age;  
    private String gender;  
    private double assignment1;  
    private double assignment2;  
    private double finalScore;  
    private static double formula1 = 0.35;  
    private static double formula2 = 0.65;
```

Figure: Class definition and Encapsulation private variables

Class definition, which is the class Student to be applied. This class is going to be used to manage data and behaviour related to the form of student in the program and data.

Encapsulation for all of the private variables. Encapsulates in the state of Student object.

```
public Student(String id, String name, int age, String gender, double assignment1,
double assignment2) {
    this.id = id;
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.assignment1 = assignment1;
    this.assignment2 = assignment2;
    calculateFinalScore();
}

public void calculateFinalScore() {
    this.finalScore = (assignment1 * formula1) + (assignment2 * formula2);
}

// Getters and setters
public String getId() { return id; }
public String getName() { return name; }
public int getAge() { return age; }
public String getGender() { return gender; }
public double getAssignment1() { return assignment1; }
public double getAssignment2() { return assignment2; }
public double getFinalScore() { return finalScore; }

public void setAssignment1(double assignment1) {
    this.assignment1 = assignment1;
    calculateFinalScore();
}

public void setAssignment2(double assignment2) {
    this.assignment2 = assignment2;
    calculateFinalScore();
}

public static void setFormula1(double newFormula1) {
    formula1 = newFormula1;
}

public static void setFormula2(double newFormula2) {
    formula2 = newFormula2;
}
```

Figure: Encapsulation for the public variables and Constructor and Methods

All of the public methods, which are the getters and setters will be used to give accessibility towards those private variables. But they are still going to be controlled and maintained.

First constructor would initialize the Student object with provided need to be filled. Another set of constructor would be the calculateFinalScore() to set the initial final score.

calculateFinalScore(): Calculates the final score based on assignments and formulas and the weights.

Setter methods for assignment1 and assignment2: Update scores and recalculate the final score.

Static variables for formula1 and formula2 are static variables shared across all Student instances. They represent weights for calculating the final score.

Static methods, setFormula1() and setFormula2(): Allow changing the formulas for all Student objects.

```
public class StudentTableModel extends AbstractTableModel {
    private Map<String, Student> students = new HashMap<>();

    public int getColumnCount() {
        return 5; // id, name, age, assignment1, assignment2
    }

    public int getRowCount() {
        return students.size();
    }

    public Object getValueAt(int row, int column) {
        String studentId = (String) students.keySet().toArray()[row];
        Student student = students.get(studentId);
        switch (column) {
            case 0:
                return student.getId();
            case 1:
                return student.getName();
            case 2:
                return student.getAge();
            case 3:
                return student.getAssignment1();
            case 4:
                return student.getAssignment2();
            default:
                return null;
        }
    }
}
```


Class definition for the StudentTable model and AbstractTable model, for creating the custom table model and displaying student's data.

This is using specific data structure, hashmap. To store objects which are the Students and to make the management much easier and efficient to be done. Student will have student ID as keys.

AbstractTableModel Methods:

getColumnCount(): Returns the number of columns in the table (5 in this case).

getRowCount(): Returns the number of rows, which is the number of students in the map.

getValueAt(int row, int column): Retrieves the value for a specific cell in the table.

```
public void addStudent(Student student) {
    students.put(student.getId(), student);
    fireTableRowsInserted(getRowCount() - 1, getRowCount());
}

public void removeStudent(String studentId) {
    students.remove(studentId);
    fireTableRowsDeleted(getRowCount() - 1, getRowCount());
}
```

Data Management methods, which is the main ones that the user will be looking at first and uses mostly. AddStudent and RemoveStudent to the model data.

```
public class GUIStudentDatabase extends JFrame {

    private static HashMap<String, Student> students = new HashMap<>();
    private DefaultTableModel tableModel;
    private JTextField searchField;
    private JTextField formula1Field;
    private JTextField formula2Field;
    private TableRowSorter<DefaultTableModel> sorter;
    // CSV File for students' data
    private static final String CSV_FILE_PATH = "students.csv";
```

Private variables that are used to create the table and setting up dataset file for storing student's database.

```
public GUIStudentDatabase() {
    // setup the main window
    setTitle("Student Database Management System");
```

```

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setSize(800, 600);
setLocationRelativeTo(null);

// create main panel for the BorderLayout()
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());

// Search bar
searchField = new JTextField();
@Override
public void insertUpdate(DocumentEvent e) {
    search();
}

@Override
public void removeUpdate(DocumentEvent e) {
    search();
}

@Override
public void changedUpdate(DocumentEvent e) {
    search();
}
});

panel.add(searchField, BorderLayout.NORTH);

// Table to display student data, only assignment1 and assignment2 are editable
String[] columnNames = {"ID", "Name", "Age", "Gender", "Assignment 1", "Assignment 2", "Final Score"};
tableModel = new DefaultTableModel(columnNames, 0) {
    @Override
    public boolean isCellEditable(int row, int column) {
        return column == 4 || column == 5;
    }
};

// add an updater or listener, whenever there is an update towards the student data
JTable table = new JTable(tableModel);
table.getModel().addTableModelListener(e -> {
    int row = e.getFirstRow();
    int column = e.getColumn();
    if (column == 4 || column == 5) {
        String id = (String) tableModel.getValueAt(row, 0);
        Student student = students.get(id);
        if (column == 4) {
            student.setAssignment1((Double) tableModel.getValueAt(row, column));
        } else {
            student.setAssignment2((Double) tableModel.getValueAt(row, column));
        }
    }
});

```

```

    }
    student.calculateFinalScore();
    tableModel.setValueAt(student.getFinalScore(), row, 6);
    }
});

// Add sorting functionality to the table
sorter = new TableRowSorter<>(tableModel);
table.setRowSorter(sorter);
table.getTableHeader().addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int column = table.columnAtPoint(e.getPoint());
        if (column >= 0) {
            boolean isAscending = sorter.getSortKeys().isEmpty() ||
                sorter.getSortKeys().get(0).getSortOrder() == SortOrder.DESENDING;
            sorter.setSortKeys(List.of(new RowSorter.SortKey(column, isAscending ? SortOrder.ASCENDING :
SortOrder.DESENDING)));
            sorter.sort();
        }
    }
});

// Add scroll pane, add table to the panel
JScrollPane scrollPane = new JScrollPane(table);
panel.add(scrollPane, BorderLayout.CENTER);

```

This is used for creating the graphical user interface, for the database management using swing. This is also the main part where all of the functionality and features are added and would start to be initiated from there.

```

JButton addButton = new JButton("Add Student");
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        addStudent();
    }
});
buttonPanel.add(addButton);

JButton removeButton = new JButton("Remove Student");
removeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        removeStudent();
    }
});
buttonPanel.add(removeButton);

```

Creating addStudent and removeStudent button in the program. ActionListener to call respective function.

```

JPanel formulaPanel = new JPanel();
formulaPanel.setLayout(new FlowLayout());

JLabel formula1Label = new JLabel("Assignment 1 Weight:");
formula1Field = new JTextField("0.35", 5);
JLabel formula2Label = new JLabel("Assignment 2 Weight:");
formula2Field = new JTextField("0.65", 5);
JButton updateFormulaButton = new JButton("Update Formula");
updateFormulaButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        updateFormula();
    }
});

formulaPanel.add(formula1Label);
formulaPanel.add(formula1Field);
formulaPanel.add(formula2Label);
formulaPanel.add(formula2Field);
formulaPanel.add(updateFormulaButton);

```

Formula update panel, as the assignment1 and assignment2 having a weight that can be changed by the teacher/user

```
private void addStudent() {
    JTextField idField = new JTextField();
    JTextField nameField = new JTextField();
    JTextField ageField = new JTextField();
    JTextField genderField = new JTextField();
    JTextField assignment1Field = new JTextField();
    JTextField assignment2Field = new JTextField();

    Object[] fields = {
        "ID:", idField,
        "Name:", nameField,
        "Age:", ageField,
        "Gender (M/F):", genderField,
        "Assignment 1 Score:", assignment1Field,
        "Assignment 2 Score:", assignment2Field
    };

    int option = JOptionPane.showConfirmDialog(this, fields, "Add Student",
        JOptionPane.OK_CANCEL_OPTION);
    if (option == JOptionPane.OK_OPTION) {
        String id = idField.getText();
        String name = nameField.getText();
        int age = Integer.parseInt(ageField.getText());
        String gender = genderField.getText();
        double assignment1 = Double.parseDouble(assignment1Field.getText());
        double assignment2 = Double.parseDouble(assignment2Field.getText());

        // Exception statement for adding student
        if (students.containsKey(id)) {
            JOptionPane.showMessageDialog(this, "Student with this ID already
exists. Please enter a different ID.");
            return;
        }

        Student student = new Student(id, name, age, gender, assignment1,
assignment2);
        students.put(id, student);
        addStudentToTable(student);
    }
}
```

This method creates a dialog for inputting student details, creates a new Student object, and adds it to the students map and table.

```
private void removeStudent() {
    String id = JOptionPane.showInputDialog(this, "Enter student ID to remove:");
    if (id != null && !id.isEmpty()) {
        if (students.containsKey(id)) {
            students.remove(id);
            removeStudentFromTable(id);
            JOptionPane.showMessageDialog(this, "Student removed successfully.");
        } else {
            JOptionPane.showMessageDialog(this, "Student not found.");
        }
    }
}
```

This method prompts for a student ID and removes the corresponding student from the map and table.

```
private void addStudentToTable(Student student) {
    Object[] rowData = {
        student.getId(),
        student.getName(),
        student.getAge(),
        student.getGender(),
        student.getAssignment1(),
        student.getAssignment2(),
        student.getFinalScore()
    };
    tableModel.addRow(rowData);
}

private void removeStudentFromTable(String id) {
    for (int i = 0; i < tableModel.getRowCount(); i++) {
        if (tableModel.getValueAt(i, 0).equals(id)) {
            tableModel.removeRow(i);
            break;
        }
    }
}
```

These methods are literally used for adding and removing Student from the table itself as from the display

```

private void search() {
    String searchText = searchField.getText().toLowerCase();
    TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<
(tableModel);
    sorter.setRowFilter(null);

    // Get all students from the HashMap and store them in a list
    List<Student> studentsList = new ArrayList<>();
    for (Student student : this.students.values()) {
        if (student.getName().toLowerCase().contains(searchText) ||
            String.valueOf(student.getAge()).contains(searchText) ||
            student.getGender().contains(searchText) ||
            String.valueOf(student.getAssignment1()).contains(searchText)
||
            String.valueOf(student.getAssignment2()).contains(searchText))
        {
            studentsList.add(student);
        }
    }

    // Update the table model with the filtered students
    tableModel.setRowCount(0);
    for (Student student : studentsList) {
        Object[] rowData = {
            student.getId(),
            student.getName(),
            student.getAge(),
            student.getGender(),
            student.getAssignment1(),
            student.getAssignment2(),
            student.getFinalScore()
        };
        tableModel.addRow(rowData);
    }
}

```

Filters the table based on the search text, matching against various student attributes.

```

private void updateFormula() {
    double formula1 = Double.parseDouble(formula1Field.getText());
    double formula2 = Double.parseDouble(formula2Field.getText());

    for (Student student : students.values()) {
        student.setFormula1(formula1);
    }
}

```

```

        student.setFormula2(formula2);
        student.calculateFinalScore();
    }

    updateTableData();
}

private void updateTableData() {
    tableModel.setRowCount(0);
    for (Student student : students.values()) {
        addStudentToTable(student);
    }
}

```

Updating the assessment weight and will automatically recalculate the final score of the students

```

private void loadDataFromCSV() {
    try (BufferedReader br = new BufferedReader(new FileReader(CSV_FILE_PATH))) {
        String line;
        boolean isFirstLine = true;
        while ((line = br.readLine()) != null) {
            if (isFirstLine) {
                isFirstLine = false;
                continue; // Skip the header line
            }
            String[] values = line.split(",");
            Student student = new Student(
                values[0], // ID
                values[1], // Name
                Integer.parseInt(values[2]), // Age
                values[3], // Gender
                Double.parseDouble(values[4]), // Assignment1
                Double.parseDouble(values[5]) // Assignment2
            );
            students.put(student.getId(), student);
            addStudentToTable(student);
        }
    } catch (IOException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error loading data from CSV file: " + e.getMessage());
    }
}

private void saveDataToCSV() {
    try (BufferedWriter bw = new BufferedWriter(new FileWriter(CSV_FILE_PATH))) {
        // Write header
        bw.write("ID,Name,Age,Gender,Assignment1,Assignment2");
        bw.newLine();
    }
}

```



```

// Write student data
for (Student student : students.values()) {
    bw.write(String.format("%s,%s,%d,%s,%.1f,%.1f",
        student.getId(),
        student.getName(),
        student.getAge(),
        student.getGender(),
        student.getAssignment1(),
        student.getAssignment2()
    ));
    bw.newLine();
}
} catch (IOException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(this, "Error saving data to CSV file: " + e.getMessage());
}
}

```

These methods are used for handling and managing the modifications and updates happening with the CSV file(student data)

Working Program

Starting menu

Assessment1 and assessment2 weights and these students are pre-set sample data.

Student Database Management System

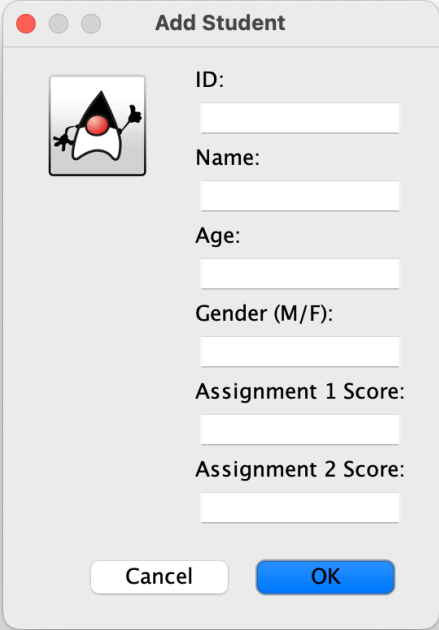
ID	Name	Age	Gender	Assignment 1	Assignment 2	Final Score
S009	David Thomas	18	M	83.0	86.0	84.949999999...
S007	Matthew Anderson	19	M	87.5	89.0	88.475
S018	Amelia Lewis	18	F	90.5	88.0	88.875
S008	Sophia Taylor	21	F	91.0	93.5	92.625
S019	Alexander Robin...	20	M	88.5	93.0	91.425
S005	Daniel Wilson	22	M	90.0	95.0	93.25
S016	Charlotte Harris	19	F	91.5	89.5	90.2
S006	Olivia Martinez	20	F	95.0	91.0	92.4
S017	Ethan Clark	21	M	84.0	92.0	89.2
S003	Michael Brown	21	M	78.0	92.5	87.425
S014	Mia Gonzalez	22	F	93.0	87.5	89.425
S004	Emma Davis	18	F	88.5	87.0	87.525
S015	William Perez	20	M	87.0	91.5	89.925
S001	John Smith	20	M	85.5	90.0	88.425
S012	Isabella Lopez	21	F	88.0	94.0	91.9
S002	Emily Johnson	19	F	92.0	88.5	89.725
S013	Benjamin Lee	18	M	85.0	90.5	88.575
S010	Ava Garcia	20	F	89.5	92.0	91.125
S021	Matthew Staniswi...	100	M	100.0	100.0	100.0
S011	James Rodriguez	19	M	92.5	88.0	89.575
S020	Harper Walker	19	F	92.0	90.0	90.699999999...

Add Student


Remove Student

Assignment 1 Weight: 0.35Assignment 2 Weight: 0.65Update Formula

id	name	gender	age	assignment 1 score	assignment 2 score
18		M		83.0	86.0
19	son	M		87.5	89.0
18					88.0
21					93.5
20	in...				93.0
22					95.0
19	s				89.5
20					91.0
21					92.0
21					92.5
22					87.5
18					87.0
20					91.5
20					90.0
21					94.0
19					88.5
18					90.5
20					92.0
100	wi...				100.0
19	ez				88.0
19					90.0



Add Student



ID:
Name:
Age:
Gender (M/F):
Assignment 1 Score:
Assignment 2 Score:

Menu screen, when user press the Add Student button. A new window open for user to fill out this form to register the student.

9
8
1
0
2
9
0
1
1
2
8
0
0
1
9
8
0
9
9

M 87.5

Add Student



ID: S021

Name: tthew Staniswinata

Age: 18

Gender (M/F): M

Assignment 1 Score: 100

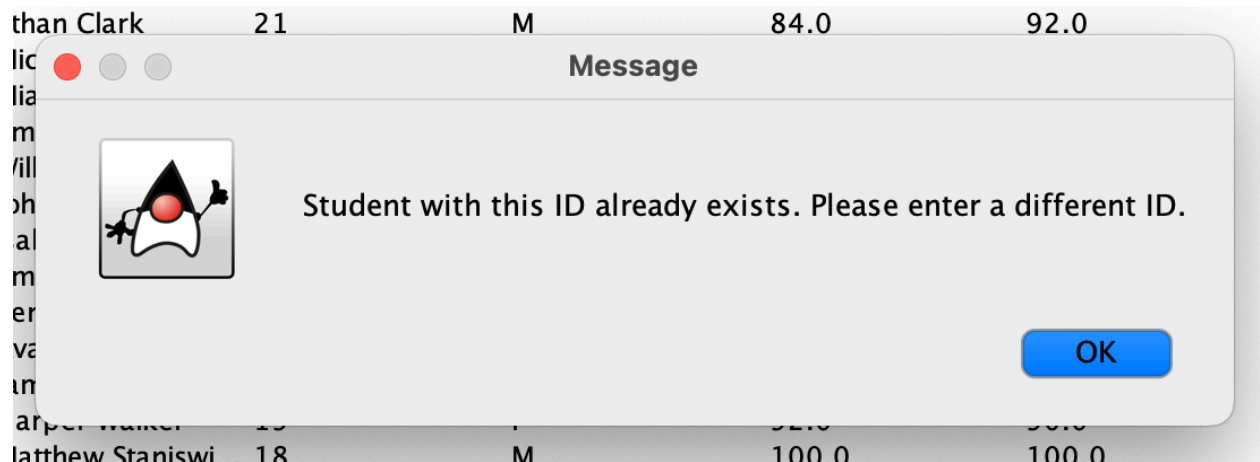
Assignment 2 Score: 100

Cancel OK

If everything is done correctly, like the example above. As everything is filled, if one of them is not filled there will cause an error.

S013	Benjamin Lee	18	M	85.0	90.5	88.575
S010	Ava Garcia	20	F	89.5	92.0	91.125
S011	James Rodriguez	19	M	92.5	88.0	89.575
S020	Harper Walker	19	F	92.0	90.0	90.6999999999...
S021	Matthew Staniswi...	18	M	100.0	100.0	100.0

Figure: Successfully add “Matthew Staniswinata” student in the data. Display in the table



Due to using hashmap, with the idea of key-value data structure.
 Every student will have the key(unique value) as the student ID. So if a student with this student ID already exist in the data, then user needs to enter a new one. As appear above the warning window.

Sorting based on final score

ID	Name	Age	Gender	Assignment 1	Assignment 2	Final Score ▾
S005	Daniel Wilson	22	M	90.0	95.0	93.25
S008	Sophia Taylor	21	F	91.0	93.5	92.625
S006	Olivia Martinez	20	F	95.0	91.0	92.4
S012	Isabella Lopez	21	F	88.0	94.0	91.9
S019	Alexander Robin...	20	M	88.5	93.0	91.425
S010	Ava Garcia	20	F	89.5	92.0	91.125
S020	Harper Walker	19	F	92.0	90.0	90.699999999...
S016	Charlotte Harris	19	F	91.5	89.5	90.2
S015	William Perez	20	M	87.0	91.5	89.925
S002	Emily Johnson	19	F	92.0	88.5	89.725
S011	James Rodriguez	19	M	92.5	88.0	89.575
S014	Mia Gonzalez	22	F	93.0	87.5	89.425
S017	Ethan Clark	21	M	84.0	92.0	89.2
S018	Amelia Lewis	18	F	90.5	88.0	88.875
S013	Benjamin Lee	18	M	85.0	90.5	88.575
S007	Matthew Anderson	19	M	87.5	89.0	88.475
S001	John Smith	20	M	85.5	90.0	88.425
S004	Emma Davis	18	F	88.5	87.0	87.525
S003	Michael Brown	21	M	78.0	92.5	87.425
S009	David Thomas	18	M	83.0	86.0	84.949999999...

Sorting based on Name

ID	Name ▲
S019	Alexander Robin...
S018	Amelia Lewis
S010	Ava Garcia
S013	Benjamin Lee
S016	Charlotte Harris
S005	Daniel Wilson
S009	David Thomas
S002	Emily Johnson
S004	Emma Davis
S017	Ethan Clark
S020	Harper Walker
S012	Isabella Lopez
S011	James Rodriguez
S001	John Smith
S007	Matthew Anderson
S014	Mia Gonzalez
S003	Michael Brown
S006	Olivia Martinez
S008	Sophia Taylor
S015	William Perez

Searching, just by looking at any of the related string. It will automatically just update the table to the student's name that has element M in it. If not then they won't be shown in the table.

M

ID	Name ▲	Age	Gender	Assignment 1	Assignment 2	Final Score
S018	Amelia Lewis	18	F	90.5	88.0	88.875
S013	Benjamin Lee	18	M	85.0	90.5	88.575
S009	David Thomas	18	M	83.0	86.0	84.949999999...
S002	Emily Johnson	19	F	92.0	88.5	89.725
S004	Emma Davis	18	F	88.5	87.0	87.525
S011	James Rodriguez	19	M	92.5	88.0	89.575
S001	John Smith	20	M	85.5	90.0	88.425
S007	Matthew Anderson	19	M	87.5	89.0	88.475
S014	Mia Gonzalez	22	F	93.0	87.5	89.425
S003	Michael Brown	21	M	78.0	92.5	87.425
S006	Olivia Martinez	20	F	95.0	91.0	92.4
S015	William Perez	20	M	87.0	91.5	89.925

Custom assessment weight based on user, as they can choose how much weight does each assessment going to be. Don't forget to press updateFormula, for the final score can be automatically recalculated again.

Here I changed from 0.35 and 0.65 respectively into 0.5 for each assessment

ID	Name ▲	Age	Gender	Assignment 1	Assignment 2	Final Score
S019	Alexander Robin...	20	M	88.5	93.0	90.75
S018	Amelia Lewis	18	F	90.5	88.0	89.25
S010	Ava Garcia	20	F	89.5	92.0	90.75
S013	Benjamin Lee	18	M	85.0	90.5	87.75
S016	Charlotte Harris	19	F	91.5	89.5	90.5
S005	Daniel Wilson	22	M	90.0	95.0	92.5
S009	David Thomas	18	M	83.0	86.0	84.5
S002	Emily Johnson	19	F	92.0	88.5	90.25
S004	Emma Davis	18	F	88.5	87.0	87.75
S017	Ethan Clark	21	M	84.0	92.0	88.0
S020	Harper Walker	19	F	92.0	90.0	91.0
S012	Isabella Lopez	21	F	88.0	94.0	91.0
S011	James Rodriguez	19	M	92.5	88.0	90.25
S001	John Smith	20	M	85.5	90.0	87.75
S007	Matthew Anderson	19	M	87.5	89.0	88.25
S014	Mia Gonzalez	22	F	93.0	87.5	90.25
S003	Michael Brown	21	M	78.0	92.5	85.25
S006	Olivia Martinez	20	F	95.0	91.0	93.0
S008	Sophia Taylor	21	F	91.0	93.5	92.25
S015	William Perez	20	M	87.0	91.5	89.25

Add Student
Remove Student

Assignment 1 Weight: 0.5
Assignment 2 Weight: 0.5
Update Formula

References

1. How to Fix Student Data Management Problems
<https://campuscafesoftware.com/how-to-fix-student-data-management-problems/>
2. Dr. Maria Seraphina Astriani, S. Kom., M. T. I.
3. Jude Joseph Lamug Martinez MCS
4. Final Project Data Structure, Matthew, Steven, Ryan,
<https://docs.google.com/document/d/1UZ6PEErNzWOIZkpKDnJUeAr3hx-WZgZBzKydL3cQgt0/edit>
5. Hashmap implementation,
<https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/>
6. Learning about the basics of JFrame,
<https://www.codejava.net/java-se/swing/jframe-basic-tutorial-and-examples>
7. Learning about the basics of Swing, <https://www.guru99.com/java-swing-gui.html>
8. Creating images to share the code by Carbon, <https://carbon.now.sh>