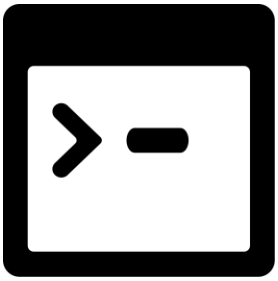


CLI Grading System

TABLE OF CONTENTS

Abstract

This document will describe and study the requested assignment that revolves around a grading system that is implemented as a command line interface, as well as inspect each class.

We will showcase a screenshot of the project itself, as well as some code snapshots. Please note that there are some comments in the code which will further aid the understanding of the workflow.

Section 1: Code Structure

The main class will simply call the initializeDB() function “to store template data” as well as the printMainMenufunction() “to print the main menu and start listening to user’s input”.

```java
public class Main {
    //Don't create a new object manually as it wont be added to the list implicitly, use addStudent/addAssignment instead
    public static void main(String[] args) {
        //Calls a function which handles students/assignments template insertions
        initializeDB();

        //Initializes the CLI by printing the main menu and waiting for an input
        printMainMenu();
    }
}
```

The feature about the whole workflow is that it heavily relies on static methods; meaning that you can immediately call any function without creating an object. An object is only created when you add a student/assignment, even this is handled internally.

The code also contains user-defined exceptions, these exceptions are set to be triggered upon the following events:

1. A user entered a wrong datatype
2. A user entered an invalid mark (not between 0 and 100)
3. A user entered a non-existing option

Please refer to the following UML diagram to further aid the code structure:

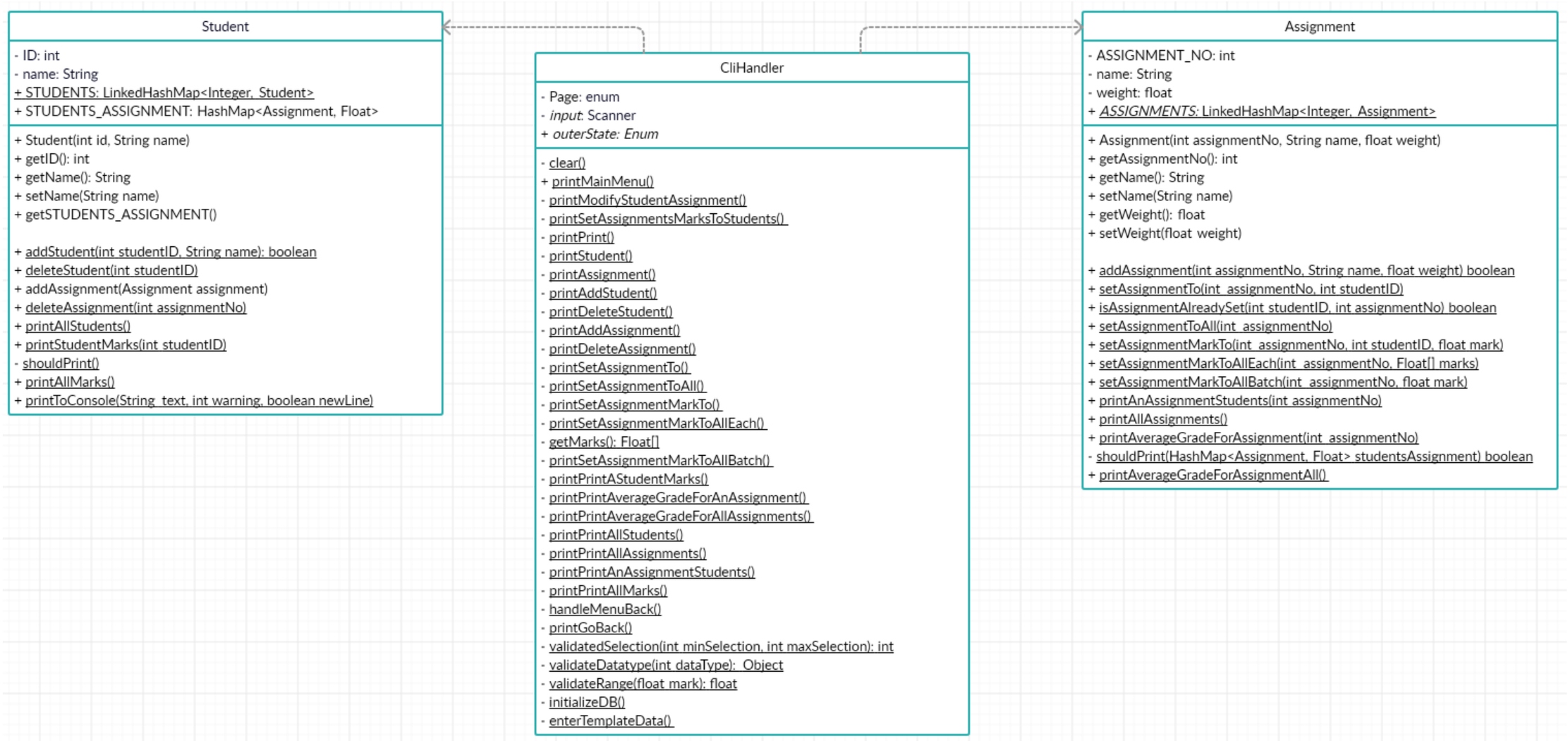


Figure 1: UML class diagram

Section 2: Student Class Overview

The student class, as any other typical class, contains variables, a parameterized constructor, as well as setters and getters.

```java
//A linked hashmap that stores the student objects
static final LinkedHashMap<Integer, Student> STUDENTS = new LinkedHashMap<>();
//The ID of the student; cant be changed
private final int ID;
//The name of the student
private String name;
//A hashmap that shows the student's registered assignment, each student object has one
private final HashMap<Assignment, Float> STUDENTS_ASSIGNMENT;

//Getter,setters

public int getID() { return ID; }

public String getName() { return name; }

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

public HashMap<Assignment, Float> getSTUDENTS_ASSIGNMENT() { return STUDENTS_ASSIGNMENT; }

////////////////////////////////////////////////////////////////////////////////

//A parameterized constructor which assigns each values respectively as well as allocates space for STUDENTS_ASSIGNMENT
Student(int ID, String name) {
    this.ID = ID;
    this.name = name;
    STUDENTS_ASSIGNMENT = new HashMap<>();
}
```

The add student function simply creates an object based on the verified function parameters, then adds it to a static hash map which is in the class as well.

```java
//Creates a new student object and add it to the hashmap, the parameters should adhere to some requirements
static boolean addStudent(int studentID, String name) {
    if (name.length() > 20)
        printToConsole( text: "The student name is too long", warning: 1, newLine: true);
    else if (STUDENTS.containsKey(studentID)) {
        printToConsole( text: "ID already exists", warning: 1, newLine: true);
        return false;
    } else {
        STUDENTS.put(studentID, new Student(studentID, name));
        printToConsole( text: "Student registered successfully", warning: 0, newLine: true);
    }
    return true;
}
```

The remove student method removes the student object from the hash map.

```
//Adds a new assignment for a student
public void addAssignment(Assignment assignment) {
    if(!STUDENTS_ASSIGNMENT.containsKey(assignment)) {
        printToConsole( text: "Assignment \"" + assignment.getName() + "\" is set to " + this.getName(), warning: 0, newLine: true);
        STUDENTS_ASSIGNMENT.put(assignment, -1.0f);
    }
    else printToConsole( text: "This assignment is already set to student "+this.getName(), warning: 1, newLine: true);
}
```

printAllStudents() method iterates through the hash map that stores the student's objects, then prints out their information.

```
//Prints a list of the students
static void printAllStudents() {
    if (!STUDENTS.isEmpty()) {
        printToConsole( text: "ID\t\tName", warning: 2, newLine: true);
        printToConsole( text: "--\t\t----", warning: 2, newLine: true);
        for (Student student : STUDENTS.values())
            System.out.println(student.getID() + "\t\t" + student.getName());
    } else printToConsole( text: "No marks to display", warning: 1, newLine: true);
}
```

printStudentMarks method simply prints out the student's information (given his ID), then prints all of his assignment marks as well as the average (the average depends on the assignment's weight).

```
//Prints the student marks in each assignment as well as adding the average
static void printStudentMarks(int studentID) {
    if (!STUDENTS.containsKey(studentID))
        printToConsole( text: "The student ID doesn't exist", warning: 1, newLine: true);
    else {
        boolean printed = false;
        float sum = 0;
        int count = 0;
        if (!STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().isEmpty() && shouldPrint(STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT())) {
            printed = true;
            printToConsole(String.format("%-20s", STUDENTS.get(studentID).getName() + "(" + STUDENTS.get(studentID).getID() + ")" + ":"), warning: 0, newLine: false);
            for (Assignment assignment : ASSIGNMENTS.values()) {
                if (STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment) != null && STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment) != -1.0) {
                    printToConsole( text: assignment.getName() + "(" + assignment.getWeight() + ")" + " ", warning: 0, newLine: false);
                    if (STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment) >= 50)
                        printToConsole(String.format("%-15s", STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment)), warning: 2, newLine: false);
                    else
                        printToConsole(String.format("%-15s", STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment)), warning: 3, newLine: false);
                    sum += STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().get(assignment) * assignment.getWeight();
                    count += assignment.getWeight();
                } else {
                    printToConsole( text: assignment.getName() + "(" + assignment.getWeight() + ")" + " ", warning: 0, newLine: false);
                    printToConsole(String.format("%-15s", "-"), warning: 1, newLine: false);
                }
            }
            printToConsole( text: "Average is: ", warning: 0, newLine: false);
            if (sum / count >= 50)
                printToConsole( text: String.format("%.2f", sum / count) + "", warning: 2, newLine: true);
            else printToConsole( text: String.format("%.2f", sum / count) + "", warning: 3, newLine: true);
        }
        if (!printed)
            printToConsole( text: "No marks to display for student " + STUDENTS.get(studentID).getName(), warning: 1, newLine: true);
    }
}
```

shouldPrint is a supplementary function that aids the previous method; it returns whether this student got his assignment graded to not (indicated if the mark is set to -1.0).

```
//Checks if the mark of the assignment is -1.0 or not "the default mark when it is not set"
private static boolean shouldPrint(HashMap<Assignment, Float> studentsAssignment) {
    for (Float mark : studentsAssignment.values())
        if (mark != -1)
            return true;
    return false;
}
```

printAllMarks() method collects reads all the students information and assignments, then orders them descending, and then, calls printStudentMarks in an iteration.

```
//Orders the students depending on their average, then prints it
static void printAllMarks() {
    HashMap<HashMap<Student, HashMap<Assignment, Float>>, Float> studentsData = new HashMap<>();
    for (Student student : STUDENTS.values()) {
        HashMap<Student, HashMap<Assignment, Float>> studentData = new HashMap<>();
        studentData.put(student, student.getSTUDENTS_ASSIGNMENT());
        studentsData.put(studentData, -1.0f);
    }
    for (HashMap<Student, HashMap<Assignment, Float>> students : studentsData.keySet()) {
        float sum = 0;
        int count = 0;
        for (HashMap<Assignment, Float> studentAssignments : students.values()) {
            for (Assignment studentAssignment : studentAssignments.keySet()) {
                if (studentAssignments.get(studentAssignment) != -1.0f) {
                    sum += studentAssignments.get(studentAssignment) * studentAssignment.getWeight();
                    count += studentAssignment.getWeight();
                }
            }
            if (!Float.isNaN((sum / count)))
                studentsData.put(students, (sum / count));
            else studentsData.put(students, -1.0f);
        }
    }
    boolean printed = false;
    if (!studentsData.isEmpty()) {
        float max = Collections.max(studentsData.values());
        while (max != -1.0f) {
            for (HashMap<Student, HashMap<Assignment, Float>> students : studentsData.keySet()) {
                if (studentsData.get(students) != null)
                    if (studentsData.get(students) == max && max != -1.0f) {
                        printStudentMarks(students.entrySet().iterator().next().getKey().getID());
                        studentsData.put(students, -1.0f);
                        students.keySet().removeIf(student1 -> student1 == students.entrySet().iterator().next().getKey());
                        max = Collections.max(studentsData.values());
                        printed = true;
                    } else if (Float.isNaN(max))
                        max = Collections.max(studentsData.values());
            }
        }
    }
    if (studentsData.isEmpty() || !printed)
        printToConsole( text: "No marks to display",  warning: 1,  newLine: true);
}
```

printToConsole customizes the console output to support selected colors depending on the parameter (does not function on windows CMD).

```
//Prints an array of colors; depending on the parameters
//Windows CMD does not support ANSI escape code, hence, the color wont change and it will print the text as is
static void printToConsole(String text, int warning, boolean newLine) {
    if (newLine)
        switch (warning) {
            case 0 -> System.out.println(text);
            case 1 -> System.out.println("\u001B[33m" + text + "\u001B[0m");
            case 2 -> System.out.println("\u001B[34m" + text + "\u001B[0m");
            case 3 -> System.out.println("\u001B[31m" + text + "\u001B[0m");
            case 4 -> System.out.println("\u001B[46m" + text + "\u001B[0m");
        }
    else
        switch (warning) {
            case 0 -> System.out.print(text);
            case 1 -> System.out.print("\u001B[33m" + text + "\u001B[0m");
            case 2 -> System.out.print("\u001B[34m" + text + "\u001B[0m");
            case 3 -> System.out.print("\u001B[31m" + text + "\u001B[0m");
            case 4 -> System.out.print("\u001B[46m" + text + "\u001B[0m");
        }
}
```

Section 3: Assignment Class Overview

Just like the student class, assignment class contains variables, a parameterized constructor, as well as setters and getters.

```
//A linked hashmap that stores the assignment objects
static final LinkedHashMap<Integer, Assignment> ASSIGNMENTS = new LinkedHashMap<>();
//The number of the assignment; cant be changed
private final int ASSIGNMENT_NO;
//The name of the assignment
private String name;
//The weight of the assignment; the higher the value, the more of the assignment can inflict on the student's average
private float weight;

//Getter,setters

public int getAssignmentNo() { return ASSIGNMENT_NO; }

public String getName() { return name; }

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

public float getWeight() { return weight; }

public void setWeight(float weight) { this.weight = weight; }

////////////////////////////////////////////////////////////////////////////////

//A parameterized constructor which assigns each values respectively
Assignment(int assignmentNo, String name, float weight) {
    this.ASSIGNMENT_NO = assignmentNo;
    this.name = name;
    this.weight = weight;
}
```

addAssignment method puts an assignment object into the hash map.

```
//Creates a new assignment object and add it to the hashmap, the parameters should adhere to some requirements
static boolean addAssignment(int assignmentNo, String name, float weight) {
    if (name.length() > 30)
        printToConsole( text: "The assignment name is too long", warning: 1, newLine: true);
    else if (ASSIGNMENTS.containsKey(assignmentNo)) {
        printToConsole( text: "Assignment number already exists", warning: 1, newLine: true);
        return false;
    } else {
        ASSIGNMENTS.put(assignmentNo, new Assignment(assignmentNo, name, weight));
        printToConsole( text: "Assignment registered successfully", warning: 0, newLine: true);
    }
    return true;
}
```

DeleteAssignment method removes an assignment object from the hashmap, also, it should remove the assignment set to the student to prevent null reference exceptions.

```
//Deletes an assignment from the list
static void deleteAssignment(int assignmentNo) {
    if (ASSIGNMENTS.containsKey(assignmentNo)) {
        ASSIGNMENTS.put(assignmentNo, null);
        ASSIGNMENTS.remove(assignmentNo);
        for (Student student : STUDENTS.values())
            student.getSTUDENTS_ASSIGNMENT().keySet().removeIf(assignment1 -> assignment1.getAssignmentNo() == assignmentNo);
        printToConsole( text: "Assignment deleted successfully",  warning: 0,  newLine: true);
    } else
        printToConsole( text: "The assignment number doesn't exist",  warning: 1,  newLine: true);
}
```

setAssignmentTo method adds an assignment object reference into the assignment hash map of a specific student.

```
//Sets a specific assignment to a specific student
static void setAssignmentTo(int assignmentNo, int studentID) {
    if (!STUDENTS.containsKey(studentID))
        printToConsole( text: "The student ID doesn't exist",  warning: 1,  newLine: true);
    else if (!ASSIGNMENTS.containsKey(assignmentNo))
        printToConsole( text: "The assignment number doesn't exist",  warning: 1,  newLine: true);
    else
        STUDENTS.get(studentID).addAssignment(ASSIGNMENTS.get(assignmentNo));
}
```

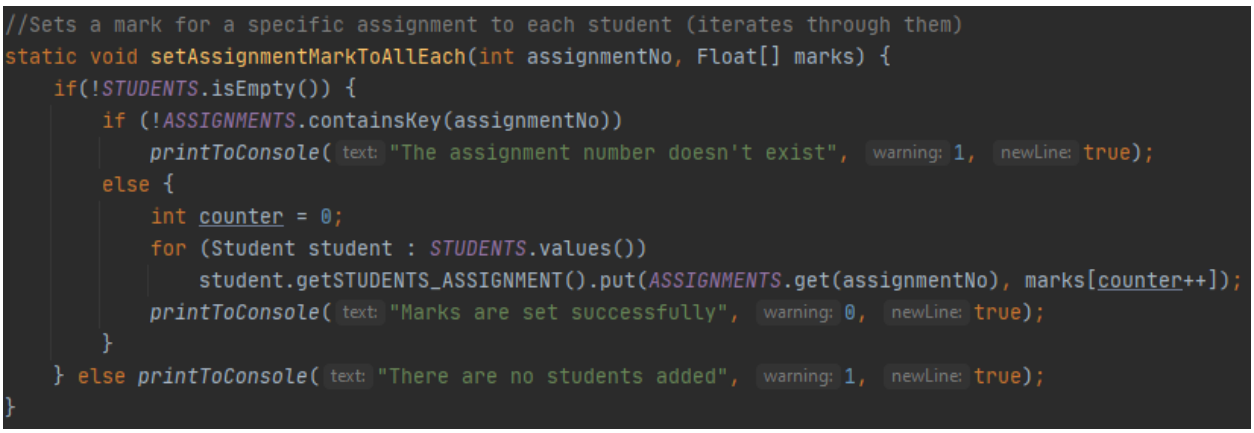
setAssignmentTo method adds an assignment object reference into the assignment hash map of all of the student.

```
//Sets a specific assignment to all students
static void setAssignmentToAll(int assignmentNo) {
   if(!STUDENTS.isEmpty()) {
       if (!ASSIGNMENTS.containsKey(assignmentNo))
           printToConsole( text: "The assignment number doesn't exist",  warning: 1,  newLine: true);
       else {
           for (Student student : STUDENTS.values())
               student.getSTUDENTS_ASSIGNMENT().put(ASSIGNMENTS.get(assignmentNo), -1.0f);
           printToConsole( text: "Assignment \"" + ASSIGNMENTS.get(assignmentNo).getName() + "\" is set to all students",  warning: 0,  newLine: true);
       }
   } else printToConsole( text: "There are no students added",  warning: 1,  newLine: true);
}
```

setAssignmentMarkTo method adds a mark to a student's assignment instead of the default mark (-1.0).

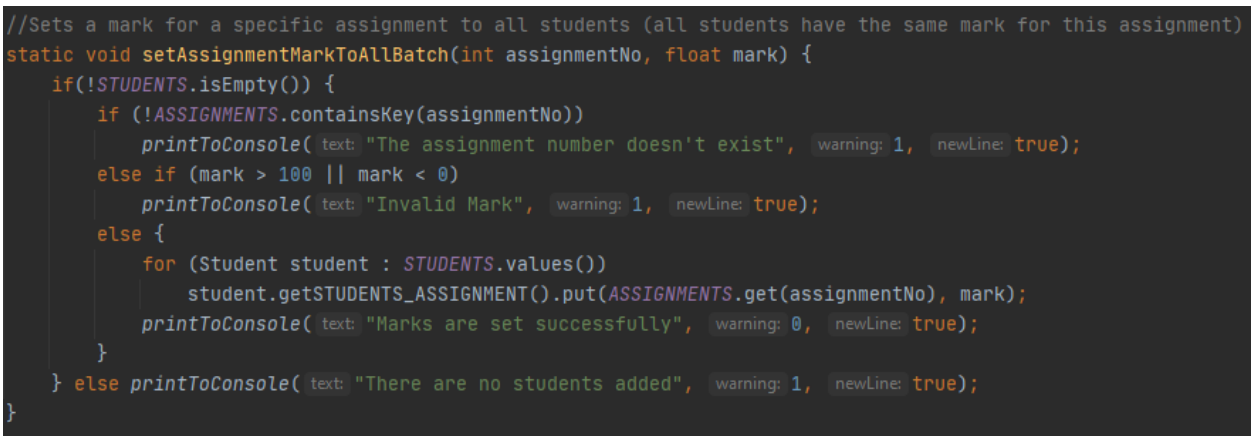
```
//Sets a mark for a specific assignment to a specific student
static void setAssignmentMarkTo(int assignmentNo, int studentID, float mark) {
    if (!STUDENTS.containsKey(studentID))
        printToConsole( text: "The student ID doesn't exist", warning: 1, newLine: true);
    else if (!ASSIGNMENTS.containsKey(assignmentNo))
        printToConsole( text: "The assignment number doesn't exist", warning: 1, newLine: true);
    else if (mark > 100 || mark < 0)
        printToConsole( text: "Invalid Mark", warning: 1, newLine: true);
    else {
        if (!STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().isEmpty() && STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().containsKey(ASSIGNMENTS.get(assignmentNo))) {
            STUDENTS.get(studentID).getSTUDENTS_ASSIGNMENT().put(ASSIGNMENTS.get(assignmentNo), mark);
            printToConsole( text: "Mark is set successfully", warning: 0, newLine: true);
        } else printToConsole( text: "Operation failed, assignment is not set to such student", warning: 1, newLine: true);
    }
}
```

setAssignmentMarkToAllEach method prompts the user to enter each student a mark for the assignment, respectively.



```
//Sets a mark for a specific assignment to each student (iterates through them)
static void setAssignmentMarkToAllEach(int assignmentNo, Float[] marks) {
    if(!STUDENTS.isEmpty()) {
        if (!ASSIGNMENTS.containsKey(assignmentNo))
            printToConsole( text: "The assignment number doesn't exist", warning: 1, newLine: true);
        else {
            int counter = 0;
            for (Student student : STUDENTS.values())
                student.getSTUDENTS_ASSIGNMENT().put(ASSIGNMENTS.get(assignmentNo), marks[counter++]);
            printToConsole( text: "Marks are set successfully", warning: 0, newLine: true);
        }
    } else printToConsole( text: "There are no students added", warning: 1, newLine: true);
}
```

setAssignmentMarkToAllBatch method is similar to setAssignmentMarkToAllEach, but it only prompts the user to enter one mark for all of the students.



```
//Sets a mark for a specific assignment to all students (all students have the same mark for this assignment)
static void setAssignmentMarkToAllBatch(int assignmentNo, float mark) {
    if(!STUDENTS.isEmpty()) {
        if (!ASSIGNMENTS.containsKey(assignmentNo))
            printToConsole( text: "The assignment number doesn't exist", warning: 1, newLine: true);
        else if (mark > 100 || mark < 0)
            printToConsole( text: "Invalid Mark", warning: 1, newLine: true);
        else {
            for (Student student : STUDENTS.values())
                student.getSTUDENTS_ASSIGNMENT().put(ASSIGNMENTS.get(assignmentNo), mark);
            printToConsole( text: "Marks are set successfully", warning: 0, newLine: true);
        }
    } else printToConsole( text: "There are no students added", warning: 1, newLine: true);
}
```

printAnAssignmentStudents simply prints the students who are participating in this assignment.

```
//Prints a list showing the students who have this assignment registered
static void printAnAssignmentStudents(int assignmentNo) {
    if (!ASSIGNMENTS.containsKey(assignmentNo))
        printToConsole( text: "The assignment number doesn't exist",  warning: 1,  newLine: true);
    else {
        ArrayList<String> studentsForThisAssignment = new ArrayList<>();
        for (Student student : STUDENTS.values()) {
            for (Assignment assignment : student.getSTUDENTS_ASSIGNMENT().keySet())
                if (assignment.getAssignmentNo() == assignmentNo)
                    studentsForThisAssignment.add(student.getName());
        }
        if (!studentsForThisAssignment.isEmpty()) {
            printToConsole(ASSIGNMENTS.get(assignmentNo).getName(),  warning: 2,  newLine: true);
            printToConsole( text: "---------------------------",  warning: 2,  newLine: true);
            for (String name : studentsForThisAssignment)
                printToConsole(name,  warning: 0,  newLine: true);
        } else
            printToConsole( text: "No students have \"" + ASSIGNMENTS.get(assignmentNo).getName() + "\" set",  warning: 1,  newLine: true);
    }
}

//Prints a list of assignments and their properties
```

printAllAssignments() shows a list of all of the assignments and their properties.

```
//Prints a list of assignments and their properties
static void printAllAssignments() {
    if (!STUDENTS.isEmpty()) {
        printToConsole(String.format("%-15s %-30s %-15s", "Number", "Name", "Weight"),  warning: 2,  newLine: true);
        printToConsole(String.format("%-15s %-30s %-15s", "----------", "--------------------------", "-------"),  warning: 2,  newLine: true);
        for (Assignment assignment : ASSIGNMENTS.values())
            printToConsole((String.format("%-15s %-30s %-15s", assignment.getAssignmentNo(), assignment.getName(), assignment.getWeight())),  warning: 0,  newLine: true);
    } else printToConsole( text: "No assignments to display",  warning: 1,  newLine: true);
}
```

As the name implies, printAverageGradeForAssignment prints the average grade of the students who have this assignment registered.

```
//Calculates the average mark for this assignment (ignores any unset mark)
static void printAverageGradeForAssignment(int assignmentNo) {
    if (!ASSIGNMENTS.containsKey(assignmentNo))
        printToConsole( text: "The assignment number doesn't exist",  warning: 1,  newLine: true);
    else {
        boolean printed = false;
        float sum = 0;
        int count = 0;
        for (Student student : STUDENTS.values()) {
            if (!student.getSTUDENTS_ASSIGNMENT().isEmpty() && student.getSTUDENTS_ASSIGNMENT().containsKey(ASSIGNMENTS.get(assignmentNo)) && shouldPrint(student.getSTUDENTS_ASSIGNMENT())) {
                printed = true;
                if (student.getSTUDENTS_ASSIGNMENT().get(ASSIGNMENTS.get(assignmentNo)) != -1.0) {
                    sum += student.getSTUDENTS_ASSIGNMENT().get(ASSIGNMENTS.get(assignmentNo));
                    count++;
                }
            }
        }
        if (!printed)
            printToConsole( text: "No students have \"" + ASSIGNMENTS.get(assignmentNo).getName() + "\" set or marks have not been set yet",  warning: 1,  newLine: true);
        else {
            printToConsole( text: "\"" + ASSIGNMENTS.get(assignmentNo).getName() + "\" With a weight of: ",  warning: 0,  newLine: false);
            printToConsole( text: ASSIGNMENTS.get(assignmentNo).getWeight() + "",  warning: 2,  newLine: false);
            printToConsole( text: " Have an average of: ",  warning: 0,  newLine: false);
            if (sum / count >= 50)
                printToConsole( text: sum / count + "",  warning: 2,  newLine: true);
            else printToConsole( text: sum / count + "",  warning: 3,  newLine: true);
        }
    }
}
```

shouldPrint is a supplementary function that aids the previous method; it returns whether this student got his assignment graded to not (indicated if the mark is set to -1.0).

```
//Checks if the mark of the assignment is -1.0 or not "the default mark when it is not set"
private static boolean shouldPrint(HashMap<Assignment, Float> studentsAssignment) {
    for (Float mark : studentsAssignment.values())
        if (mark != -1)
            return true;
    return false;
}
```

printAverageGradeForAssignmentAll calls printAverageGradeForAssignment

in a loop for each assignment.

```
//Calls printAverageGradeForAssignment() multiple times to show all the assignment's average
static void printAverageGradeForAssignmentAll() {
    if(!ASSIGNMENTS.isEmpty())
    for (Assignment assignment : ASSIGNMENTS.values())
        printAverageGradeForAssignment(assignment.getAssignmentNo());
    else printToConsole( text: "No assignments to display",  warning: 1,  newLine: true);
}
```

Section 4: CliHandler Class Overview

The CliHandler class first begins which an enum class declaration, as well as variables declared to aid the class to function.

```
private enum Page {
    MainMenu,
    Modify_Student_Assignment,
    Set_Assignments_Marks_To_Students,
    Print,
    PrintStudent,
    PrintAssignment
}

//A scanner to log user's commands
private static final Scanner input = new Scanner(System.in);
//An enum to know which menu the user is currently in
private static Enum<Page> outerState;
```

clear() method simply prints multiple new lines to simulate a ‘cls’ command.

```
//Stimulates clearing the screen by printing many new lines
private static void clear() { printToConsole( text: "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n", warning: 0, newLine: true); }
```

printMainMenu() is one of many methods that shows the user a menu and prompts him to enter a value. Depending on the value entered, the program goes to a new menu or opens a function.

```
static void printMainMenu() {
    clear();
    printToConsole( text: "Main Menu", warning: 4, newLine: true);
    printToConsole( text: "Please Select An Option..\n", warning: 0, newLine: true);
    printToConsole( text: "1. Modify Student/Assignment", warning: 0, newLine: true);
    printToConsole( text: "2. Set Assignments/Marks To Students", warning: 0, newLine: true);
    printToConsole( text: "3. Print", warning: 0, newLine: true);
    printToConsole( text: "0. Exit", warning: 1, newLine: true);

    //Calls a function to display the next menu depending on the user's selected option
    outerState = Page.MainMenu;
    switch (validatedSelection(1, 3)) {
        case 1 -> printModifyStudentAssignment();
        case 2 -> printSetAssignmentsMarksToStudents();
        case 3 -> printPrint();
        case 0 -> System.exit( status: 0);
    }
}
```

printAddStudent() is one of many methods that requests the user a group of requirements; in order to fulfill a functionality, then, it will navigate into the previous menu.

```
private static void printAddStudent() {
    clear();
    printToConsole( text: "Please enter the ID followed by the name of the student",  warning: 0,  newLine: true);
    int studentID = (int) validateDatatype(0);
    String studentName = (String) validateDatatype(2);
    if (!addStudent(studentID, studentName)) {
        printToConsole( text: "Overwrite existing data? (Y/N)",  warning: 0,  newLine: true);
        String response = (String) validateDatatype(2);
        while (!(response.toLowerCase().equals("y") || response.toLowerCase().equals("n"))) {
            printToConsole( text: "Invalid input",  warning: 1,  newLine: true);
            response = (String) validateDatatype(2);
        }
        if (response.toLowerCase().equals("y")) {
            STUDENTS.get(studentID).setName(studentName);
            printToConsole( text: "Student registered",  warning: 0,  newLine: true);
        }
    }
    printGoBack();
}
```

handleManuBack() simply oversees the current state of the user “i.e. whether if he is in the main menu or in the print menu”, and then prints the previous menu.

```
//A function which modifies the enum depending on the current menu state when back is triggered
private static void handleMenuBack() {
    if (outerState == Page.MainMenu)
        printMainMenu();
    else if (outerState == Page.Modify_Student_Assignment) {
        outerState = Page.MainMenu;
        printModifyStudentAssignment();
    } else if (outerState == Page.Set_Assignments_Marks_To_Students) {
        outerState = Page.MainMenu;
        printSetAssignmentsMarksToStudents();
    } else if (outerState == Page.Print) {
        outerState = Page.MainMenu;
        printPrint();
    } else if (outerState == Page.PrintStudent) {
        outerState = Page.Print;
        printStudent();
    } else if (outerState == Page.PrintAssignment) {
        outerState = Page.Print;
        printAssignment();
    }
}
```

printGoBack() prints a back option after an operation is finished, then goes back whenever the user inputs the correct value (which is 0).

```
//A function which modifies the enum depending on the current state when a function is finished
private static void printGoBack() {
    printToConsole( text: "\n0. Back",  warning: 1,  newLine: true);
    if (validatedSelection(0, 0) == 0)
        handleMenuBack();
}
```

validateSelection prompts the user to enter the correct option range (given in the parameters), and then returns the selected value if it was valid. If not, it will keep prompting the user to enter the value until it is valid.

```
//Checks the user's option input in a menu; whether it is in the numbers range or not, continues requesting until the value is correct
private static int validatedSelection(int minSelection, int maxSelection) {
    int selection = -1;
    while (!(selection >= minSelection && selection <= maxSelection) && selection != 0) {
        try {
            selection = input.nextInt();
            if (!(selection >= minSelection && selection <= maxSelection) && selection != 0) {
                throw new OutOfBoundOptionException("Invalid input");
            }
        } catch (OutOfBoundOptionException e) {
            printToConsole( text: "Invalid number",  warning: 1,  newLine: true);

        } catch (Exception e) {
            printToConsole( text: "Invalid input",  warning: 1,  newLine: true);
            input.nextLine();
        }
    }
    return selection;
}
```

validateDataType is the similar to validateSelection, but this method will check if the datatype passed is the same as the requested datatype.

```
//Checks the user's datatype input; whether it is integer, float ,or string, continues requesting until the value is correct
private static Object validateDatatype(int dataType) {
    Object value = null;
    while (value == null) {
        try {
            if (dataType == 0)
                value = input.nextInt();
            else if (dataType == 1)
                value = input.nextFloat();
            else if (dataType == 2)
                value = input.next();
            if (value == null)
                throw new DatatypeMismatchedException("Invalid input");
        } catch (Exception e) {
            printToConsole( text: "Invalid input",  warning: 1,  newLine: true);
            input.nextLine();
        }
    }
    return value;
}
```

validateMark is also similar to the two previous methods, but it will keep checking if the mark is valid or not (between 0 and 100).

```
//Checks the user's mark input; whether its between 0 or 100 ,continues requesting until the value is correct
private static float validateRange(float mark) {
    while (mark < 0 || mark > 100) {
        try {
            if (mark > 0 || mark < 100)
                throw new InvalidMarkRange( errorMessage: "Invalid mark");
        } catch (Exception e) {
            printToConsole(e.getMessage(), warning: 1, newLine: true);
            mark = (float) validateDatatype(1);
        }
    }
    return mark;
}
```

enterTemplateData and initializeDB are set to fill the array lists/hash maps with predefined data so the program can be assessed without entering the data everytime on runtime.

```
//A function that fills the hashmaps with template data as well as manipulates them (Can be removed safely)
static void initializeDB() {
    enterTemplateData();
    setAssignmentTo( assignmentNo: 4, studentID: 2);
    setAssignmentTo( assignmentNo: 4, studentID: 1);
    setAssignmentTo( assignmentNo: 3, studentID: 2);
    setAssignmentToAll(3);
    setAssignmentToAll(2);
    setAssignmentMarkTo( assignmentNo: 4, studentID: 1, mark: 60);
    setAssignmentMarkTo( assignmentNo: 4, studentID: 2, mark: 90);
    setAssignmentMarkTo( assignmentNo: 3, studentID: 2, mark: 49);
    setAssignmentMarkTo( assignmentNo: 3, studentID: 1, mark: 80);
    setAssignmentMarkToAllBatch( assignmentNo: 4, mark: 80);
    setAssignmentMarkToAllBatch( assignmentNo: 2, mark: 75);
}

//A function that fills the hashmaps with template data
private static void enterTemplateData() {
    addStudent( studentID: 1, name: "Ahmad");
    addStudent( studentID: 2, name: "Mohammad");
    addStudent( studentID: 3, name: "Hamed");
    addStudent( studentID: 4, name: "Tamer");
    addStudent( studentID: 5, name: "Hamza");
    addStudent( studentID: 6, name: "Yousef");
    addStudent( studentID: 7, name: "Ali");
    addStudent( studentID: 8, name: "Ahmad");


    addAssignment( assignmentNo: 1, name: "Java Basics", weight: 2);
    addAssignment( assignmentNo: 2, name: "XML Basics", weight: 1);
    addAssignment( assignmentNo: 3, name: "Firebase DB", weight: 3);
    addAssignment( assignmentNo: 4, name: "Firebase Authentication API", weight: 5);
}
```

Section 5: Further Notes

There are a few notes that should be covered before diving into the code.

- The reason that the UML class diagram does not have any inheritance/ composition characteristics; is because that there was no need for them; for they may further increase the complexity of the structure.

- The user is not expected to edit the main class, he have to call only the printMainMenu() method “initializeDB() is optional”; this is because that making a new object of student/assignment will not make an effect to the output, since the object is not added to the hash map in the constructor instantly, there is a method to handle it; however, accessing any static method will yield an effect, thus, acceptable.

- The user is given the option to overwrite a student/assignment, simply add a new student/assignment with the same ID/number and preferred attributes, and then press “Y” to overwrite, or “N” to cancel. Rest assured; the students will not lose their designated assignments by doing so.

- There will be an indication whether the student has failed an assignment or not (indicated by the color of the mark, blue denotes that the student passed while red indicates that the student has failed).

- There is a length limit for the name of the student/assignment; twenty characters for the student name while thirty characters for the assignment name.

- The program should not throw any unhandled exceptions.

- The program is not suited to perform properly under Windows CMD; for it does not support ASNI escape codes. It will function, but it may print undesirable escape codes; to resolve this, please modify the “printToConsole” method at the student class or import a library which detects the operating system environment.