C - Development

Technique: Abstraction

Class: Topic, Class, and Student

Success Criteria
D. User can add and modify classes
C. User can add and modify students
E. User can add and modify topics

Justification:
The Topic, Class, and Student classes were created with relevant data so that the user can add new students, classes, and topics to the system and store their data.

Explanation:
I used abstraction to capture only the necessary data that the client needed in the system.

Code sample:

```
public class Class {

    String studentsGrade; //Refers to grade level of class(As in the students in class are studying in grade 11 or 12, etc.)
    float avgGrade;
    String avgAlphaGrade;
    String extraInformationOnClass;
    String className;
    int classID;

    ArrayList<ClassTopic> topicsStudied = new ArrayList<ClassTopic>();
    ArrayList<Integer> studentsInClass = new ArrayList<Integer>();
```

Technique: Inheritance (with abstract classes)

Class: Topic, StudentTopic, and ClassTopic

Success Criteria
E. User can add and modify topics

Justification:
The StudentTopic and ClassTopic inherit from the superclass Topic so that code duplication is reduced between the two classes.

Explanation:
The Topic class is an abstract class, an instance of it cannot be created. Instead only instances of its subclasses StudentTopic and ClassTopic can be created. This protects unwanted objects from being created (instances of superclass only).

Code sample:

```
public abstract class Topic {

    private int topicID;
    private String topicName;

public class ClassTopic extends Topic {

    private float avgGrade;
    private String avgAlphaGrade;
    private String extraNotes;

public class StudentTopic extends Topic {

    private int grade;
    private String alphaGrade;
    private String extraRemarks;
```

Technique: Encapsulation

Class: Student, Class, and Topic

Success Criteria
D. User can add and modify classes
C. User can add and modify students
E. User can add and modify topics

Justification:
The Student, Class, and Topic classes are all encapsulated to ensure that variables can only be accessed and modified through predefined methods (accessors and mutators).

Explanation:
Variables in the three classes can only be accessed through the predefined methods and as such are secured from inappropriate access or modification. For example, ID in the Student Class cannot be modified once an instance of the Student object is created. This is to ensure nobody can modify the ID once created.

Code sample:

```
public abstract class Topic {

    private int topicID;
    private String topicName;

    public Topic (int topicID, String topicName) {
        this.topicID = topicID;
        this.topicName = topicName;
    }

    public int getTopicID() {
        return topicID;
    }

    public void setTopicID(int topicID) {
        this.topicID = topicID;
    }

    public String getTopicName() {
        return topicName;
    }

    public void setTopicName(String topicName) {
        this.topicName = topicName;
    }

}
```

Technique: Data Structures (Array Lists)

Class: StudentController and ClassController

Success Criteria
A. User can search for a specific student by name and Student ID number
B. User can search for a specific class by name and Class ID number

Justification:
The ArrayLists in the StudentController and ClassController classes hold all the student and class objects in the system. This allows the client to search and sort through the students/classes.

Explanation:
The ArrayLists hold multiple instances of an object (Student and Class in this case). It allows for them to be easily stored and manipulated.

Code sample:

```
public class StudentController {
    public static ArrayList<Student> students = new ArrayList<Student>();

public class ClassController {
    public static ArrayList<Class> classes = new ArrayList<Class>();
```

Technique: Menus (Switch case)

Class: MainController and StudentController and ClassController

Success Criteria
F. User can successfully exit the system
K. User can easily navigate the system

Justification:
The client can easily navigate the system and exit menus and the system itself through the use of menus that utilize switch case statements.

Explanation:
Switch case statements allow for the user to choose specific options in a menu by entering letters to choose different options in a menu that the switch case statements then handle (Drien Vargas).

Code sample:

```
do{
    System.out.println("---------CLASSES--------");
    System.out.println(
              "[ L ] List all classes\n"
            + "[ S ] Search (And edit/view/remove) classes\n"
            + "[ O ] Sort classes\n"
            + "[ A ] Add a class\n"
            + "[ Q ] Go back to previous menu (main menu)");
    menuChoice = IBIO.inputChar("");
    menuChoice = Character.toLowerCase(menuChoice);
    switch(menuChoice){
        case 'l':
            ClassController.listAll();
            do{
                System.out.println("Once you are done viewing list, please enter P to proceed");
                menuChoice = IBIO.inputChar("");
                menuChoice = Character.toLowerCase(menuChoice);
            }while(menuChoice != 'p' && menuChoice != 'P');
            break;
        case 's':
            ClassController.searchClassMenu();
            break;
        case 'o':
            ClassController.sortClassMenu();
            break;
        case 'a':
            ClassController.addClass();
            break;
    }
}while(menuChoice != 'q' && menuChoice != 'Q');
```

Technique: File management (Load/Save)

Class: StudentController and ClassController

Success Criteria
G. User can save the information
H. User can load information

Justification:
FileManagement in the StudentController and ClassController allows the data that the client inputs in the system to be saved to a file and loaded later on when the system is launched again. This is essential to making the system usable as it is mostly useless without the ability to save data.

Explanation:
The load and save methods in the ClassController and StudentController allow the data to be saved and loaded whenever the methods are referenced in the code. The methods utilize many built-in java classes like File, FileReader, and others (Drien Vargas).

Code sample:

```
public static void load() throws IOException{
    File studentsFile = new File("students.txt");
    if (studentsFile.exists()){
        FileReader fr = new FileReader(studentsFile);
        BufferedReader load = new BufferedReader(fr);
        int studentCount = 0;
        if(load.ready()){
            studentCount = Integer.parseInt(load.readLine());
        }
        for(int i = 0; i < studentCount; i++){
            if(load.ready()){
                String name = load.readLine();
                int classID = Integer.parseInt(load.readLine());
                int studentID = Integer.parseInt(load.readLine());
                String generalRemarks = load.readLine();
                int topicCount = Integer.parseInt(load.readLine());
                ArrayList<StudentTopic> topicsToAdd = new ArrayList<StudentTopic>();
                for(int j = 0; j < topicCount; j++){
                    int topicID = Integer.parseInt(load.readLine());
                    String topicName = load.readLine();
                    int grade = Integer.parseInt(load.readLine());
                    String extraRemarks = load.readLine();
                    StudentTopic temp = new StudentTopic(topicID, topicName, grade, extraRemarks);
                    topicsToAdd.add(temp);
                }
                Student temp = new Student(name, topicsToAdd, generalRemarks, studentID, classID);
                students.add(temp);
            }
        }
        load.close();
    }
}
```

Technique: Searching (Linear Search)

Class: StudentController and ClassController

Success Criteria
A. User can search for a specific student by name and Student ID number
B. User can search for a specific class by name and Class ID number

Justification:
Search methods in the StudentController and ClassController classes allow the client to search for a Class or Student by name or ID.

Explanation:
The methods for search loop through the ArrayLists for students and classes and find the one the user is searching for (if it exists). This is known as the linear search algorithm (Drien Vargas).

Code sample:

```
public static int searchByID(int idToSearch){
    //For loop iterates through all students in students ArrayList and returns index of student in list if found
    for(int i = 0; i < students.size(); i++){
        if(students.get(i).getStudentID() == idToSearch){
            return i; //returns index of found student
        }
    }
    return -1; //returns -1 if student not found
}
```

Technique: Sorting (BubbleSort algorithm)

Class: StudentController and ClassController

Success Criteria
L. User can sort classes and students in the system

Justification:
There are multiple private methods in the StudentController and ClassController classes that are called on from a sort menu in which the user can select by what attribute he/she wants to sort by. The user can then choose whether to sort in ascending order or descending order.

Explanation:
Each private method uses the same concept which is that of the Bubble Sort algorithm. Adjacent items in the ArrayList are compared in reference to the attribute being sorted for (e.g. in reference to average grade). They are then switched if not in the correct order. The algorithm loops through this process until no more changes are required (Drien Vargas).

Code sample:

```
private static void sortByIdAscending(ArrayList<Student> studentsToSort) {
    int studentNumber = studentsToSort.size();
    boolean swapped = true;
    while(swapped){
        int pointer = 0;
        swapped = false;
        while(pointer < studentNumber - 1){
            if(studentsToSort.get(pointer).getStudentID() > studentsToSort.get(pointer + 1).getStudentID()){
                Student temp = studentsToSort.get(pointer);
                studentsToSort.set(pointer, studentsToSort.get(pointer+1));
                studentsToSort.set(pointer + 1, temp);
                swapped = true;
            }
            pointer++;
        }
        studentNumber--;
    }
}
```

Technique: Overloading

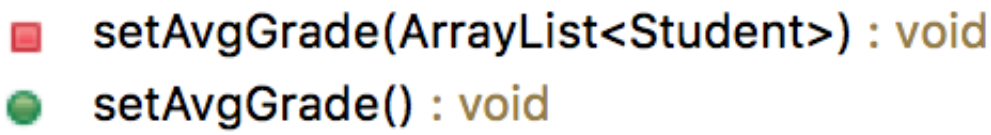
Class: Class

Success Criteria
I. User can compute average (mean) easily for specific students, classes, topics, and grades

Justification:
The overloading of methods allows multiple methods with the same name. For the Class class this is essential because it allows the correct computation of average grades even if an ArrayList of students is not provided to the class. It does this by searching the classes ArrayList in ClassController for the relevant students and retrieving their grades to compute the average.

Explanation:
While the methods may be numerous in nature and share a same name, what separates them is the method signature which is defined by the method name and the number, type, and order of its parameters (Leahy).

Code sample:



Technique: Validation (while loops)

Class: StudentController and ClassController

Success Criteria
J. User can demonstrate an error if out of bounds information is entered (e.g. number grade is 16, out of the accepted range of 1-12).

Justification:
Stops the client from entering invalid input. This stops the system from recording and saving invalid data (attributes like grade average have to be within a range).

Explanation:
Through the use of while loops, the client is repeatedly asked to re-enter information if the client has entered an invalid input.

Code sample:

```
boolean notUnique = false;
int classID = IBIO.inputInt("Enter class' ID: ");
if(classID < 0){
    notUnique = true;
}
for(int j = 0; j < classes.size(); j++){
    if(classes.get(j).getClassID() == classID)
        notUnique = true;
}
while(notUnique){
    notUnique = false;
    classID = IBIO.inputInt("Enter class' ID (Previous entry already exists in system or is invalid): ");
    if(classID < 0){
        notUnique = true;
    }
    for(int j = 0; j < classes.size(); j++){
        if(classes.get(j).getClassID() == classID)
            notUnique = true;
    }
}
```

Technique: Polymorphism (ToString Method)

Class: Class

Success Criteria
L. User can sort classes and students in the system

Justification:
When sorting and viewing classes, the Class ToString method is called. This makes it easy to print out all the classes for the client to view them.

Explanation:
The ToString method is an existing method for all Objects. It is overrode by the ToString method in Class because Class is subclass of Object (Drien Vargas).

Code sample:

```
@Override
public String toString() {
    if(this.getAvgGrade() == -1.0){
        return "Class ID: " + this.getClassID() + "\nClass name: " + this.getClassName() + "\nStudents in class: "
                + this.studentsInClass.size() + "\nClass average grade: Not enough topics to display" +
                "\nClass average alphanumerical grade: Not enough topics to display";
    }else{
        String avgGradeFormat = String.format("%.2f", this.getAvgGrade());
        return "Class ID: " + this.getClassID() + "\nClass name: " + this.getClassName() + "\nStudents in class: "
                + this.studentsInClass.size() + "\nClass average grade: " + avgGradeFormat +
                "\nClass average alphanumerical grade: " + this.getAvgAlphaGrade();
    }
}
```

Word Count: 803