CS496 Gradebook Software

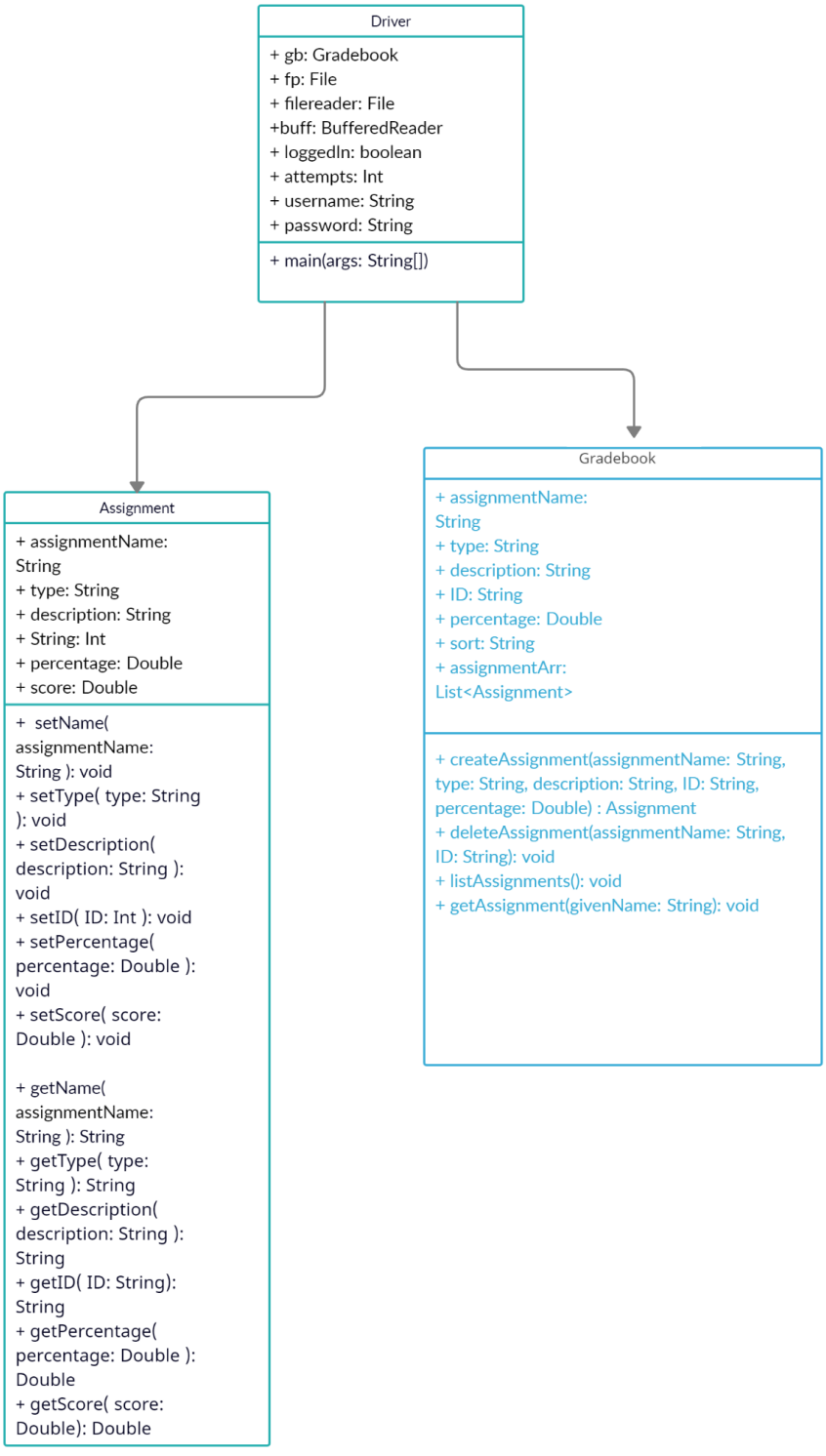
Hashim Abdirahim

Alberto Guadiana

Jason Songvilay

Software Design 2.0

1. **Updated Software Design Specification:**



Classes

The current implementation of our gradebook software consists of three classes: our Gradebook, Assignment, and Driver. Our Driver class would begin by prompting the user to login with specified credentials, to which a user has three tries to login before the program terminates. If the user logins successfully, they are greeted by the initial dialogue to choose a specific command to operate the Gradebook. Our initial implementation had much of it's logic contained into a main class, but after revisions we decided to outsource it towards multiple different classes not only for organizational purposes, but for efficiency as well.

The Gradebook class itself possesses functions designed to work on a Gradebook object created in the Driver. We aimed to provide the basic functionality that was listed in our specification document that included creating new assignments with specific attributes, deleting assignments, sorting assignments based on an assignment's percentage or name, and editing assignment fields and saving their values.

The next class we implemented was our Assignment class that held all of the specified attributes that are prompted when a user creates an assignment in the terminal. It remained virtually the same as our initial implementation, with traditional getters and setters to allow alterations to it's type, description, ID number, percentage and score. It must be noted that we did change the data types of some attributes to better fit the software usage, which will be covered in the attributes section of our document.

We did not implement our AssignmentDatabase and GradeStats classes at this time to focus on refining the required specifications outlined. Despite this, we aim to add them as additional features for our next software release. We intend for these classes and their implementations to remain the same, with AssignmentDatabase storing a list of Assignments by year and semester and GradeStats providing statistics on grades such as the average, minimum, and maximum scores from the Gradebook's assignment list.

Attributes

The attributes of our Driver class consisted of two Strings named username and password, both used for the purposes of login authentication and are currently predetermined by the developer. Two backend variables, File filereader, and BufferedReader buff are also used to compare the username and password just entered to the login information that is saved within the Gradebook software. Next, our Driver class includes variables to confirm the user has logged in successfully with boolean loggedIn and a counter Int attempts that will hold the amount of times a login has been unsuccessful. Finally, our Driver class is the primary area we create Gradebook objects in order to use the functionalities of our Gradebook, with Gradebook gb being prior to login.

Our Gradebook class utilizes the variables passed in from our Driver to create Assignments and add them to our specialized ArrayList<Assignment> assignmentArr in order to sort them by name or percentage, depending on the value of the attribute String choice. This class also performs operations involving attributes assignmentName (String), type (String), description (String), ID (String), and percentage (Double).

Some of the attributes of the Assignment class were changed since our initial implementation, with ID becoming a String rather than an Int in order for us to properly sort our Assignments by their ID. Other than this change, our Assignment class still includes assignmentName (String), type (String), description (String), percentage (Double), and score (Double). These attributes are passed from the Gradebook class when an assignment is formally created and are customizable through the basic getters and setters within this class.

Operations

Our Driver class contains a main function to begin the initialization of the console based interface. In this main function, the user is prompted to enter their username and password which are compared to a saved .txt file in our repo containing the proper login credentials in the form of "user pass". The information in the .txt file is then split by their spaces in order to compare the values of username and password individually. If they match, the user is granted access into the software. If not, the user has 3 additional chances to enter the correct credentials before the program terminates itself. On a successful login, the user is prompted with the various commands to create, delete, show, or edit assignments. Currently all of our commands do support error catching, as the user will either be: prompted again until they enter the correct value required (ex: when creating an assignment, a user will be prompted again if they enter letters in the ID field) or fall into a catch statement (ex: if the user enters a command that is not supported). When the user is completed with their work they can properly save and exit by typing

“exit”, which will terminate the program and save to another .txt containing information from their latest session.

The functions of our Assignment class have remained unchanged since our initial design. This class still consists of basic getters and setters to allow for manipulation of the attributes inputted by the user. Each parameter has a set method to alter parameters, as well as a get method to return the desired attribute.

Our Gradebook class contains all of the methods used to create, delete, edit, and list all of the assignments within the Gradebook. In our createAssignment function, we take in the assignmentName, type, description, ID, and percentage that the user enters and reprint it in the console, followed by actually creating the Assignment object and adding it to our ArrayList of Assignments assignmentArr for easy sorting later.

Next, in our deleteAssignment function, we take in the name of the assignment to be deleted along with the ID (in case assignments are named the same) and proceed into the deletion process as long as the current size of the assignmentArr is greater than zero. We check if the previous parameters match and if so, remove it from our ArrayList. After removing the assignment we print out the current status of the assignmentArr so the user can confirm what assignments still exist in their gradebook, or print out that the Gradebook is empty.

In our next function, listAssignments, we take in no parameters but instead prompt the user to type the sorting method they would like to display their assignments, either by name or by percentage. After doing so, we proceed by creating a Comparator of type Assignment to compare the values of an Assignment object's name or percentage to pass into the built in sort of the Collections package. After the list of assignments is sorted, we print them out for the user to view.

Lastly, in our editAssignment method, we take in an assignmentName and check if there is currently an assignment with the given name in the list. If so, we proceed to a while loop that prompts the user to specify which Assignment field they wish to edit. In this method we take in the user's input and convert it to lowercase automatically to avoid any ASCII errors regarding capitalization. When a specific field is specified, we call upon that specific setter from the Assignment class and change the field. For Strings, a similar process is repeated for the other fields. However, for IDs, as they must be only contained to numbers, we call upon a helper method, called isOnlyDigits, to compare if the newly entered ID matches the regex of any amount of digits from 0 to 9 before finally setting it in the Assignment. The user is able to continuously edit fields until they are finished and may complete the editing process by typing done, in which their newly set fields will be outputted in the console for them.

In all of our current implementations most of our traversing is accomplished using linear for loops, however we do acknowledge that this efficiency could be refactored and improved upon in the future release of our software.

Timeline

4/23-4/30 - Add additional classes (GradeStats, AssignmentDatabase) and refractor code if possible

4/30-5/7 - Conduct testing of new classes and refactored code and verify functionality, complete final report

1. **Verification and Validation Test Plans:** This section lays out test plans for *both* verification and validation. You are required to include tests for at least two out of the three granularities discussed in class (unit, functional, system). Be as detailed as possible, identifying which sections of your design you are testing, what the test sets/vectors are, etc. (Hint: use or modify your design diagram to indicate the target and scope of the tests, what kind of failures you are covering, etc.). Feel free to use any methods discussed in class.**Unit Test for insert Assignment:**

- When logIn = true, insert("HW1", "Math", "Math HW",001, 75)
    - Should return true boolean
    - Should print insert is successful
    - The database should contain a record of an assignment with the name HW1
- When logIn = true, insert(" ", "Math", "Math HW", 001, 75)
    - Should return false boolean
    - Should print insert was unsuccessful due to empty parameter

    - When logIn = true, insert(“HW1 ”, “Math”, “Math HW”, 001, 75)
        - Should return false boolean
        - Should print insert was unsuccessful due to assignmentId already exist
- When logIn = false, insert(“HW1”, “Math”, “Math HW”,001, 75)
    - Should return false boolean
    - Should print insert can’t be completed due Account logIn being false

Unit Test for delete Assignment:

- When logIn = true, insert(“HW1”,001)
    - Should return true
    - Should print assignment has been removed
- When logIn = true, insert(“HW1”,001)
    - Should return false
    - Should print no such assignment exist
- When logIn = true, insert(“ ”,001)
    - Should return false
    - Should print “one or more parameters is missing”

Unit Test for edit Assignment:

Assuming HW1 Still exist

- When logIn = true, edit(“HW1”)
    - Should prompt user what to edit
    - If the response does not equal either “name” or “percentage” or “id” or “type” the system will print an error message
- When logIn = true, edit(“HW3”)
    - Should return false
    - The system will prompt a message saying “ No such assignment exists”

Unit Test for sort/list Assignment:

Assume the database contains ten assignments: HW1… HW10

- When logIn = true, list(“name”)
    - Should return true
    - Should print assignment from HW1 through HW10
    - Should print “sorting successful”
- When logIn = true, list(“percentage”)
    - Should return true
    - Should print Assignment in order with the lowest percentage to highest
    - Should print “sorting successful”

Assume the database contains no assignments

- When logIn = true, list(“name”)

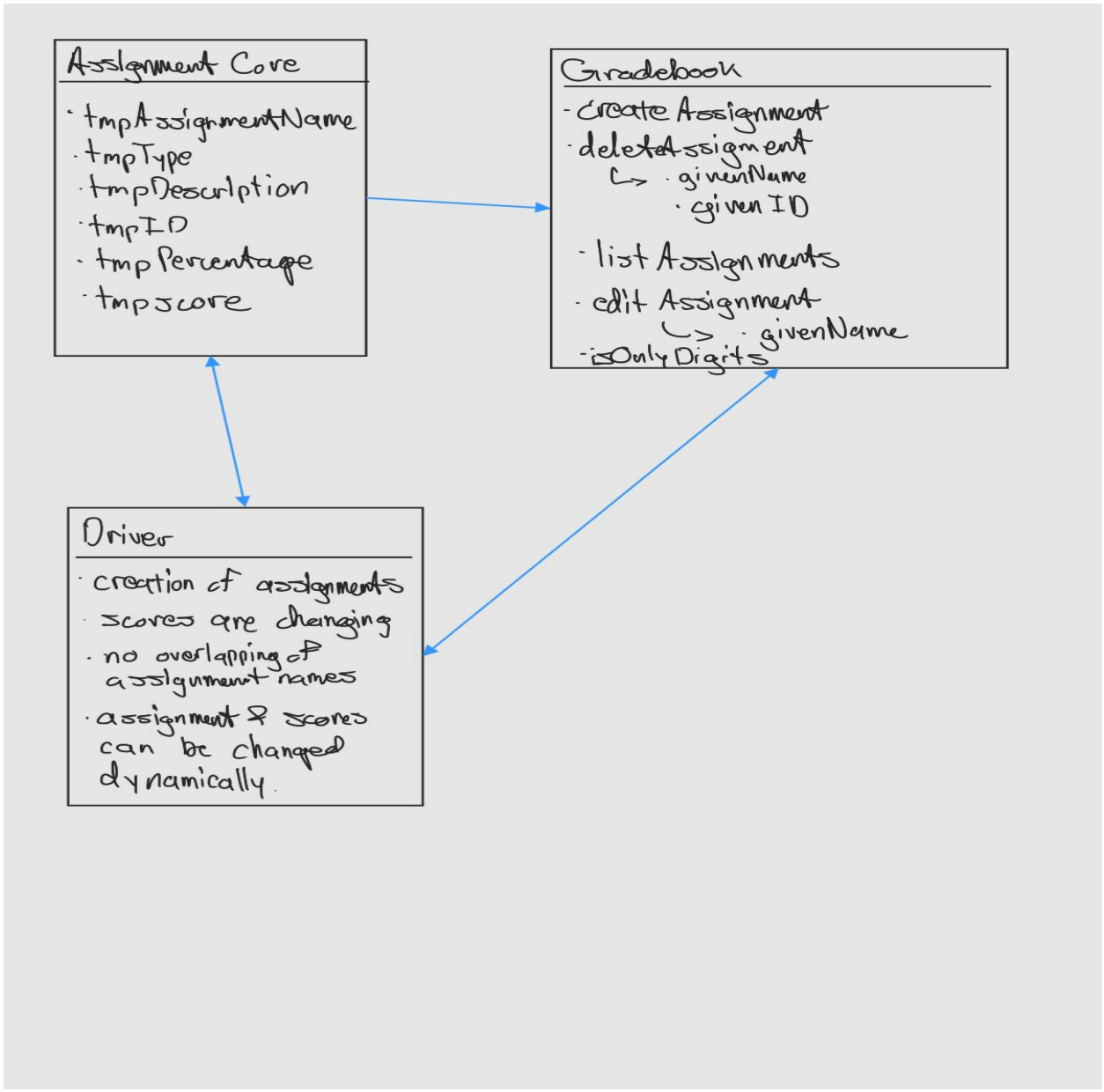
    - Should return false
    - Should print " No assignments to sort"
- When logIn = true, list("percentage")
    - Should return false
    - Should print " No assignments to sort"

System Test:

For the system test, we want to test anything that can make the system fail and prevent flaws that later affect our clients. We want to first test out the user login function. This function checks if the username/password is correct and returns either true or false. This basically gives the customer access to the program without the user or pass, the program will not move forward. To test this, we tried different usernames and passwords to see if it will grant access to the wrong account. Going forward, the user will be prompted to choose which actions he will like to complete. We then test each function rigorously by adding and deleting assignments to see if the database will give us an error. Once the function was tested, we wanted to see if the actions were saved into the database. We quit the grade book and restarted to test the assignments were still there. We kept trying to do these actions over and over to find any flaws and fix them to meet the client's requirements.

2. **Data Management Strategy:** All of your software systems revolve around persistent and potentially sensitive data. Here you should specify your data management strategy, SQL or non. (Hint: diagrams always help).

Data Management Strategy


Assignment Core
· tmpAssignmentName
· tmpType
· tmpDescription
· tmpID
· tmpPercentage
· tmpScore
Gradebook
- Create Assignment
· deleteAssignment
↳ · givenName
· given ID
· list Assignments
· edit Assignment
↳ · givenName
- isOnlyDigits
Driver
· creation of assignments
· scores are changing
· no overlapping of assignment names
· assignment & scores can be changed dynamically.