Markbook Project

Stage 1: Analysis

1A. Analysing the Problem & Criteria for Success

The key issue to be resolved is the lack of efficiency for teachers to be able to grade, input, and distribute marks for their classes. For example, the TDSB uses a system in which teacher's grade assessments, uploading the marks to a database system, which calculates the student's mark in addition to categorical marks. However, this information is exclusive to teachers and only disclosed to students when the teacher wishes. This prevents students from being able to receive their marks whenever they wish, as students are left in the dark concerning the mark.

Other school districts allow students to receive their marks online whenever they want. Due to this, these students have the advantage of knowing their mark at any time, allowing them to be able to review certain areas they struggle at or relieve some of the stress of waiting for marks.

The database system planned is an efficient school class database system with uses for students and teachers. From the teacher's side, they are able to administer marks, review entries, etc. from one program. From the student's side, they are able to view their mark breakdowns left by the teacher. Using this system, we use a database which students and teachers are able to be more efficient with marking and distribution.

The goals of for our markbook database is to be able to:

- **Teacher capabilities:**
  - Store, access, add, and delete student information.
  - Search for students information/marks by student number.
  - Sort student information by marks.
  - Sort student information by student number.
  - Display class rankings.
  - In addition, teachers have access to all student features.
  - Teachers are able to switch between courses they are teaching. (e.g ICS4U1-11 and ICS4U-15)
- **Student capabilities:**
  - Access history of student's past test/assignments/quiz marks and each category. (K/U, APP, COMM, TIPS)
  - Students are able to switch between courses they are taking. (e.g MHF4U1-14 and ICS4U1-15)

1B. Prototype Solution

When the program runs, it will begin by launching a login/sign-up page for students/teachers who will have different dashboards.

If a new student wants to sign-up:

- Prompted to enter name, course code, student #, to create account.
- Teacher teaching the course code has to approve request to permit the student to join the course
- If more than one teacher teaches a course, they pick the teacher
- Once student is apart of a class/course, they will be able to view the student dashboard.

- Properties of dashboard explained in student login section.

If a new teacher wants to sign-up:

- Prompted to enter teacher name, teacher number, and the course that they will teach,
- Will have access to the teacher dashboard.
- Once they login, they are able to:
    - Create a new course.
    - Add students.
    - Find students.
    - Delete students.
    - Sort students.
    - Add and delete entries.

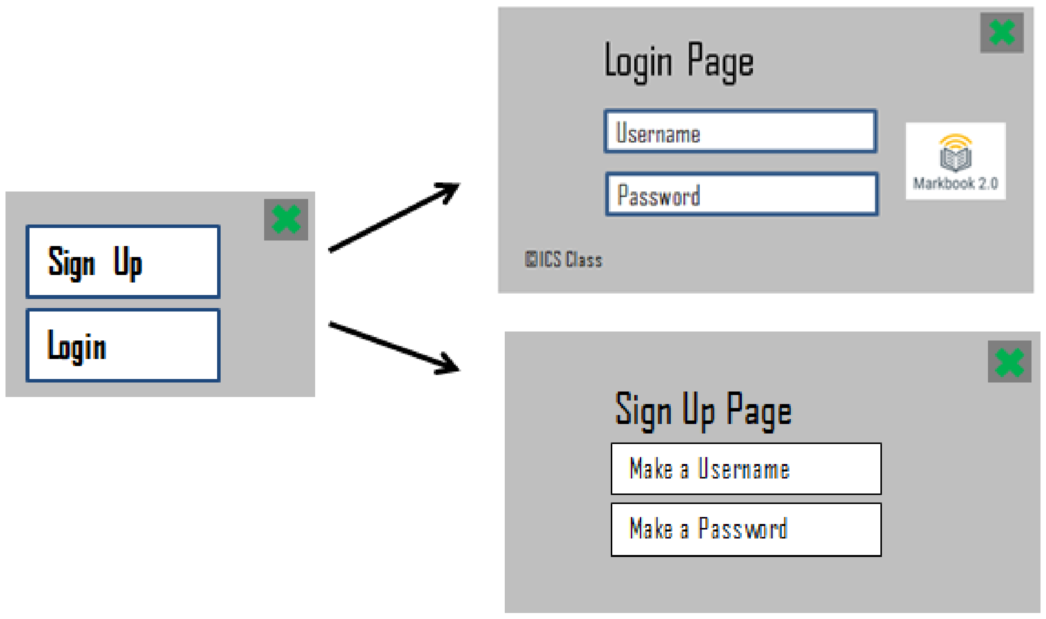
If a student wants to login:

- Must enter correct user/password (input)
- Once they login, they will be presented with a dashboard with a variety of options such as:
    - View mark breakdown.
    - View current mark.

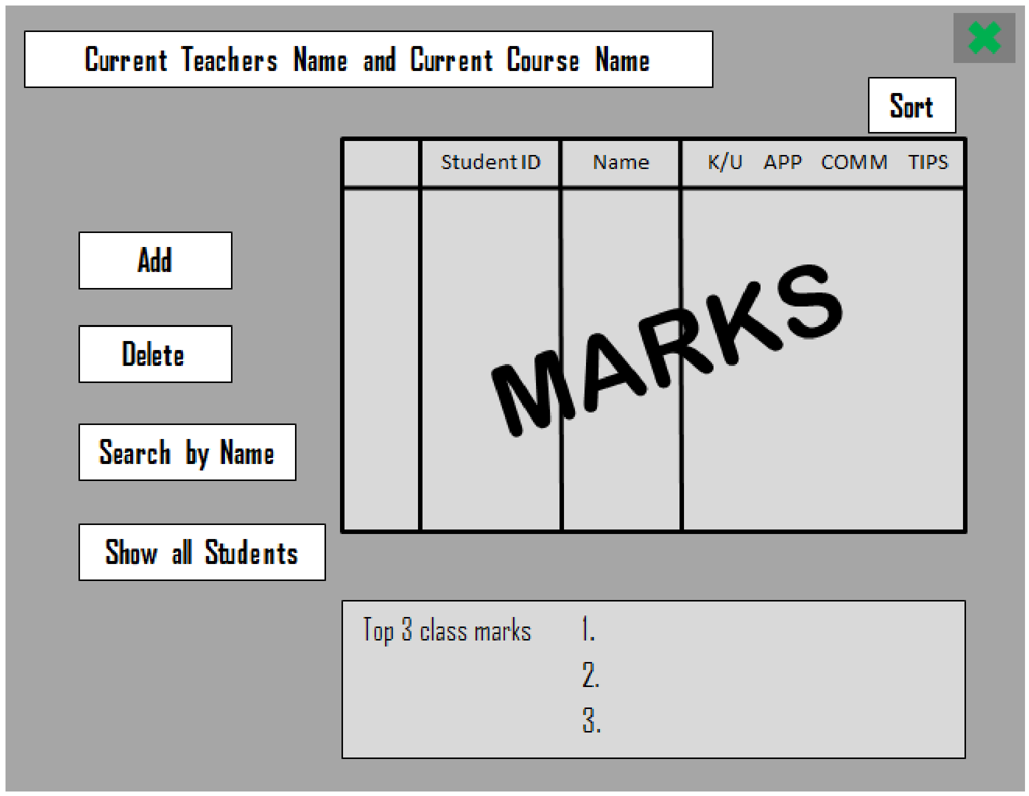
If a teacher wants to login:

- Prompted to input username and password/
- Presented with teacher dashboard that allows them to select a class they teach, depending on which class they select, a list of students in that class will be available. The teacher can select each individual student and input/delete/update marks, view mark history etc.

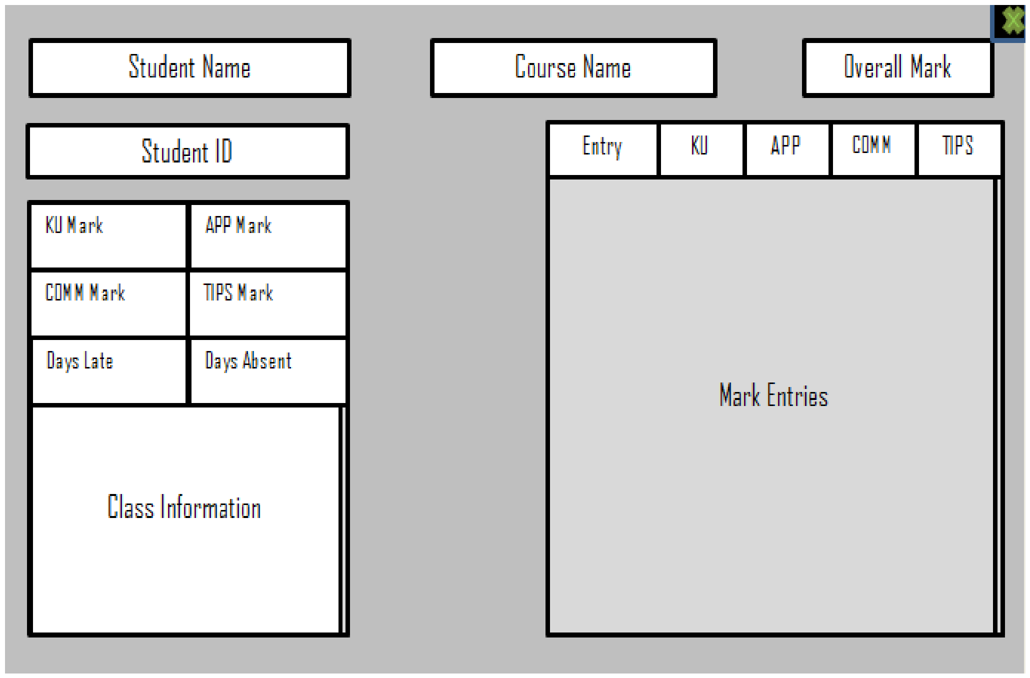
First Page :



Teacher dashboard:



Student dashboard:



Process from the start:

1. Sign-up/Login option window
2. If user chooses Sign-up, it prompts the Sign-up page. If user chooses Login, it prompts the Login Page.
3. On the Sign-up page, the user needs to create a username and password which automatically gets stored.
4. On the Login Page, the user needs to enter their username and password. If it meets a teacher's information, the teacher's dashboard will open. If it meets a students information, the student dashboard will open.
5. It's the teacher's choice to provide data to the students.
6. Students can only access the data provided by the teacher.
7. By closing the window, the user automatically gets logged out, and has to login again if they wish.

Stage 2: Design the Program

2A. Data Structures

Format of input

The database will be stored using individual txt files. Each file will contain individual file will contain all of the information pertaining to a particular student in a course. In addition, another txt file will store user information, acting as a database.

Data structure:

- Markbook
    - data
        - ICS4U1
            - ICS4U1.txt
            - 325247039.txt
        - MDM4U1
            - MDM4U1.txt
            - 323456798.txt
    - users.txt

Record data structure:

- Markbook
    - Courses:
        - ICS4U1
            - Students
                - Mark Entries
        - MDM4U1
    - Users:
        - 325247039
        - 222999
        - 321823999

Format of output

The information will be output to users using a GUI displaying the data. This allows students to easily access their information and efficiently search through the information. In addition, the user will be able to save the information to a file they can print out.

Class definitions

- **Object:** The extensive superclass.
- **Markbook:** The class storing all the information of teachers, students, and courses.
    - Datafields:
        - private ArrayList<Student> courses: A list containing all the courses in the markbook.
        - private ArrayList<Person> users: A list containing all the current users of the markbook.
    - Methods:
        - public Markbook() : Creates a new markbook. Loads all information.
        - public boolean addCourse(String course, int period, int teacherNumber): Creates a new course.
        - public boolean addEntry(String course, int period, int stuNum, double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weighting, String entry): Adds a new entry to a particular student in a specified course.
        - public boolean addStudent(String name, int studentNumber, String password): Adds a new student to the user database.
        - public boolean addStudentToCourse(String course, int period, int studentNumber): Adds a student to a specified course.
        - public boolean addTeacher(String name, int studentNumber, String password): Adds a new teacher to the user database.
        - public boolean deleteStudent(String course, int period, int stuNum): Removes a student from a course.
        - public int findUserByID(int userID): Finds user by identification number in the list of users.
        - public int getCourse(String course, int period, int index): Finds course by name and period in the list of courses.
        - public void loadData(): Loads all course data.
        - public void loadUsers(): Loads all user data.
        - public void loginUser(int userID, String pass): Performs a login attempt.
        - public void outputStudentInfo(int stuNum): Outputs all information relevant to a logged in student.
        - public void outputTeacherInfo(int teachNum): Outputs all information relevant to a logged in teacher.
        - public void saveCourses(): Saves all courses in the database.
        - public void saveUsers(): Saves all users in the database.
- **Person**: The abstract class representing all accounts within the application.
    - Datafields:
        - private String name: The individual's name.
        - private int identificationNumber: The individual's identification number used to log in.
        - private String password: The individual's password used to log in.
    - Methods:
        - public Person (String nm, int num, String pass): Creates a new person.
        - public String getName(): Returns the individual's name.

        - public int getNumber(): Returns the individual's identification number.
        - public String getPassword(): Returns the individual's password.
        - public boolean login(int num, String pass): Checks if input credentials match saved credentials.
        - public abstract void loadOutput(ArrayList<Course> courses): Outputs a user's information.
- **Teacher:** The class representing a teacher's account in the application. Extends from Person.
    - Methods:
        - public Teacher(String name, int num, String pass): Creates a new teacher account.
        - public void loadOutput(ArrayList<Course> courses): Outputs a teacher's courses.
- **Student:** The class representing a student's account in the application. Inherits from Person.
    - Methods:
        - public Student(String name, int num, String pass): Creates a new student account.
        - public void loadOutput(ArrayList<Course> courses): Outputs a student's courses.
- **Course:** The class representing an individual course accessible to both students and teachers.
    - Datafields:
        - public String course: The course code. (e.g ICS4U1-15)
        - public int periodNumber: The period number.
        - private ArrayList<StudentCourseInfo> students: The array of students taking the course.
        - private int teacherNum: Stores the teacher identification number of the teacher teaching the course.
    - Methods:
        - public Course(String courseName, int teachNum, int periodNum): Creates a new course.
        - public void addMark(int stuNum, double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weighting, String entry):Adds a new entry for a student.
        - public boolean addStudent(int studentNumber): Adds a new student to the course.
        - public boolean checkTeacher(int periodNum, int teachNum): Checks if a given teacher teaches the course.
        - public boolean checkStudent(int studentNum, int index): Checks if a given student is in the course.
        - public boolean courseOutputStudent(int studentNum, String name): Outputs student information for the course.
        - public boolean courseOutputTeacher(int teacherNum, String name): Outputs all the information for the course.
        - public boolean deleteEntry(int stuNum, String entryName): Deletes a mark entry.
        - public boolean deleteStudent(int stuNum): Removes a student from the course.
        - public ArrayList<StudentCourseInfo> getStudents(): Returns the ArrayList of students in the course.
        - public int getTeacherNumber(): Returns the teacher's identification number.
        - public void saveCourse(): Saves all the course information to the database.
        - public void sortDecreasing(): Sorts the students from highest to lowest average.
        - public void sortIncreasing(): Sorts the students from lowest to highest average.
        - public void sortDecreasingStudentNum(): Sorts the students from highest to lowest student number.

        - public void sortIncreasingStudentNum(): Sorts the students from lowest to highest student number.
- **StudentCourseInfo:** The individual student information for each student enrolled in a course.
    - Datafields:
        - private int studentNumber: Stores the student identification number.
        - private ArrayList<MarkEntry> entries: Stores all mark entries for the student.
    - Methods:
        - public StudentCourseInfo(int stuNum): Constructor for a new student.
        - public boolean addMark(double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weighting, String entry):Adds a new entry for the student.
        - public boolean checkStudent(int stuNum): Checks if the input student number matches stored student number.
        - public boolean deleteEntry(String entryName): Deletes a mark entry.
        - public double getAverage(): Calculates the average of the given individual considering the weight factor for KU, App, Comm, Tips.
        - public int getStudentNumber(): Accessor for the student number.
- **MarkEntry:** An individual mark entry for a student.
    - Datafields:
        - private double knowledge: Stores the knowledge mark for each assessment.
        - private double application: Stores the application mark for each assessment.
        - private double communication: Stores the communication mark for each assessment.
        - private double thinking: Stores the thinking and inquiry mark for each assessment.
        - private double knowledgeTotal: Stores the total knowledge marks for each assessment.
        - private double applicationTotal: Stores the total application marks for each assessment.
        - private double communicationTotal: Stores the total communication marks for each assessment.
        - private double thinkingTotal: Stores the total thinking marks for each assessment.
        - private int weight: Stores the weight for each individual assessment.
        - private String entryName: Stores the name of the assessment.
    - Methods:
        - public markEntry(double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weighting, String entry): Constructor for a mark entry.
        - public double getKnowledge(): Accessor for knowledge mark.
        - public double getApplication(): Accessor for application mark.
        - public double getCommunication(): Accesssor for communication mark.
        - public double getThinking(): Accessor for thinking mark.
        - public double getKnowledgeTotal(): Accessor for knowledge mark total.
        - public double getApplicationTotal(): Accessor for application mark total.
        - public double getCommunicationTotal(): Accessor for communication mark total.
        - public double getThinkingTotal(): Accessor for thinking mark total.
        - public int getWeight(): Accessor for weight.
        - public String getEntryName(): Accessor for entry name.
        - public double calculateKnowledge(): Calculates the knowledge average.
        - public double calculateApplication(): Calculates the application average.

        - public double calculateCommunication(): Calculates the communication average.
        - public double calculateThinking(): Calculates the thinking average.
        - public double calculateMark(): Calculates the mark of the entry.

2B. Algorithms

void saveCourse(): Saves course information.

- Pseudo code:
    - Creates and saves files with appropriate
    - Create directory with course name.
    - Create a text file with course information.
    - Create a text file with student course information for every student, repeating for every student.

void loadData(): Loads all course and student data.

- Pseudo code:
    - Go into data directory.
    - Go into first directory.
    - Find the course information file, creating a course.
    - Read all student information files, adding students with information to the course.

StudentCourseInfo loadStudent(File text): Loads all student data.

- Pseudo code:
    - Calculate the number of lines in the file.
    - Calculate the number of entries in the file.
    - Read the information appropriately, adding entries to the student information.
    - Return the student information.

void loadUsers(): Loads all user data.

- Pseudo code:
    - Go to user file.
    - Check if the first user is a teacher or student.
    - Read the following information, saving to appropriate user type.
    - Continue until all users in file are loaded.

void saveUsers(): Saves all user data:

- Pseudo code:
    - Create user file.
    - Get first user, write the user type.
    - Then write name, password, and identification number.
    - Repeat for all users.

Sorting algorithms:

- Sorting from lowest to highest average and highest to lowest average.
    - Bubble sort used, as datasets such as mark entries have a small standard distribution. (e.g. a 90 average student will have marks around 85-95%
    - Easy to implement.
    - Small datasets typically.
- Sorting from lowest to highest student number and highest to lowest student number.

    - Selection sort used, as it has few comparisons and is faster than bubble sort.
    - Easy to implement.

Mutators and Accessors:
- Mutate and access private variables.
- Used as data is encapsulated.
- Helps maintain privacy and security.

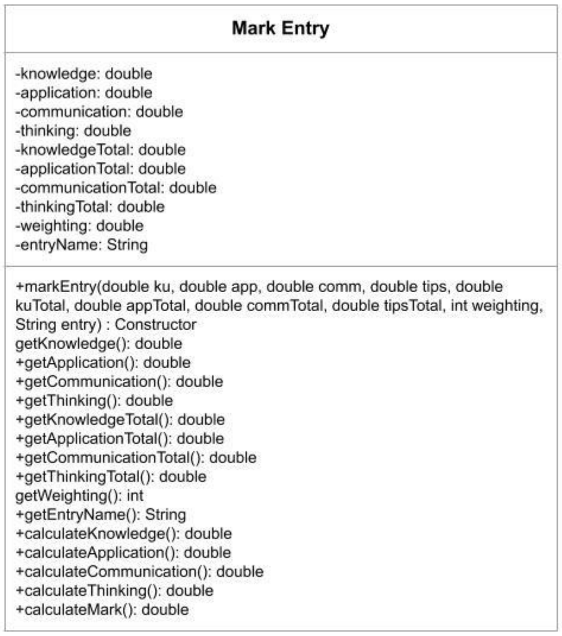
GUI Algorithms:
- Displays outputs for students and teachers by iterating through course data.

Searching algorithms:
- Algorithm returns the index of a given student in the student list or an error value if the student does not exist.
- Algorithm returns the index of a user in the user list or an error value if the user does not exist.
- Algorithm returns the index of a course in the course list or an error value if the course does not exist.

Validators:
- Checks if a teacher teaches a certain course and returns a boolean value.
- Checks if a student number is contained within the course list and returns a boolean value.
- Algorithm searches for a mark entry, deleting if it exists and returns boolean if it does not.
- Algorithm searches for a mark entry, returning false if an entry exists with the same name and adding the entry if it does not exist.
- Algorithm searches for a student, returning false if the student exists and adding a new student if the student does not exist yet.
- Algorithm searches for a student in the student list, deleting the student from a course, or returning false if student does not exist.
- Algorithm checks if a teacher exists in the database and returns false if the teacher exists or adds a new teacher.
- Algorithm checks if a teacher exists in the database and returns false if the teacher exists or adds a new teacher.
- Algorithm checks if a student exists in the database and returns false if the teacher exists or adds a new teacher.
- Algorithm checks if a student number entered does not belong to a teacher, then adds a student to a course if and only if the student does not exist.
- Algorithm adds a course to the markbook, returning false if the course already exists.
- Algorithm checks if login credentials match a certain user in the database, redirecting them to the appropriate dashboard or returning a false value if credentials are not valid.

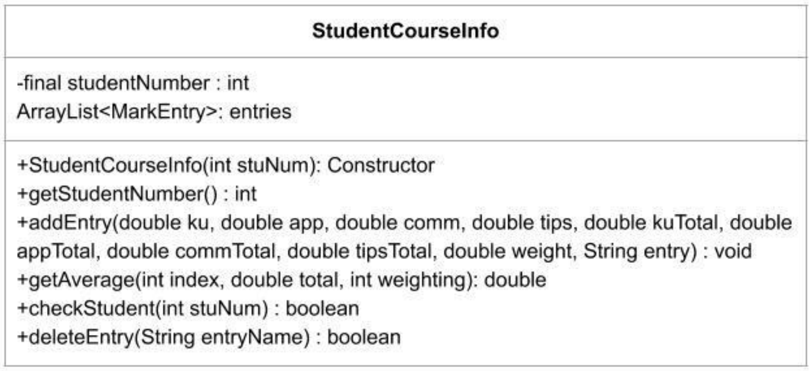
Calculators:
- Determine the mark of a certain assessment or individual.
- Category calculator determines average per assessment category.
- Assessment calculator determines average for the assessment.
- Average calculator determines average overall.

2C. Modular Organization


Mark Entry

```
-knowledge: double
-application: double
-communication: double
-thinking: double
-knowledgeTotal: double
-applicationTotal: double
-communicationTotal: double
-thinkingTotal: double
-weighting: double
-entryName: String
```

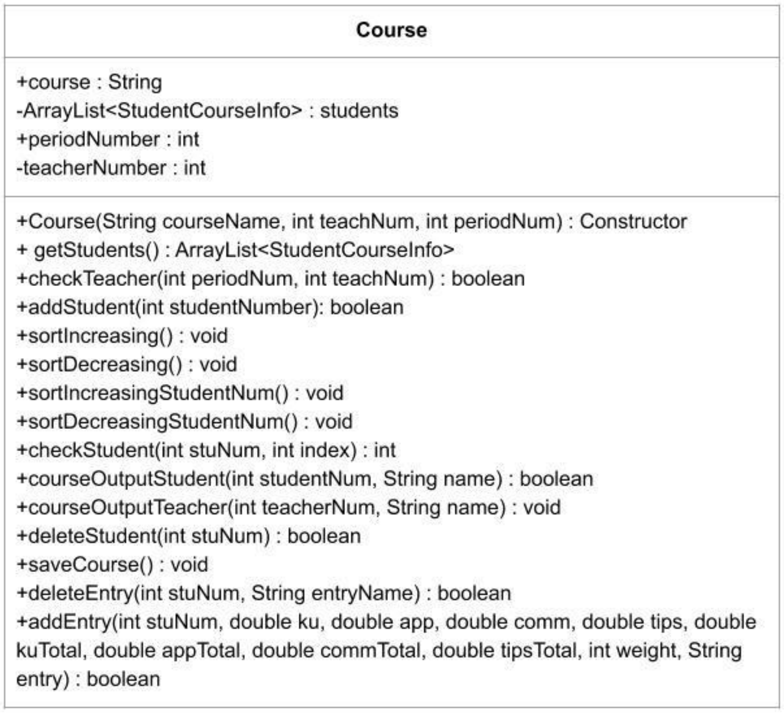
```
+markEntry(double ku, double app, double comm, double tips, double
kuTotal, double appTotal, double commTotal, double tipsTotal, int weighting,
String entry) : Constructor
getKnowledge(): double
+getApplication(): double
+getCommunication(): double
+getThinking(): double
+getKnowledgeTotal(): double
+getApplicationTotal(): double
+getCommunicationTotal(): double
+getThinkingTotal(): double
getWeighting(): int
+getEntryName(): String
+calculateKnowledge(): double
+calculateApplication(): double
+calculateCommunication(): double
+calculateThinking(): double
+calculateMark(): double
```



StudentCourseInfo

-final studentNumber : int
ArrayList<MarkEntry>: entries

+StudentCourseInfo(int stuNum): Constructor
+getStudentNumber() : int
+addEntry(double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, double weight, String entry) : void
+getAverage(int index, double total, int weighting): double
+checkStudent(int stuNum) : boolean
+deleteEntry(String entryName) : boolean

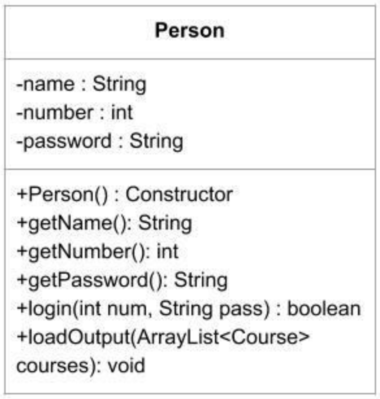


Course

+course : String
-ArrayList<StudentCourseInfo> : students
+periodNumber : int
-teacherNumber : int

+Course(String courseName, int teachNum, int periodNum) : Constructor
+ getStudents() : ArrayList<StudentCourseInfo>
+checkTeacher(int periodNum, int teachNum) : boolean
+addStudent(int studentNumber): boolean
+sortIncreasing() : void
+sortDecreasing() : void
+sortIncreasingStudentNum() : void
+sortDecreasingStudentNum() : void
+checkStudent(int stuNum, int index) : int
+courseOutputStudent(int studentNum, String name) : boolean
+courseOutputTeacher(int teacherNum, String name) : void
+deleteStudent(int stuNum) : boolean
+saveCourse() : void
+deleteEntry(int stuNum, String entryName) : boolean
+addEntry(int stuNum, double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weight, String entry) : boolean

UML DIAGRAM FOR MARKBOOK CODE (Culminating - Adrian, Yash, Kiran)



```
Person
-name : String
-number : int
-password : String
+Person() : Constructor
+getName(): String
+getNumber(): int
+getPassword(): String
+login(int num, String pass) : boolean
+loadOutput(ArrayList<Course>
courses): void
```

UML DIAGRAM FOR MARKBOOK CODE (Culminating - Adrian, Yash, Kiran)

Teacher
+Teacher(String name, int num, String pass) : Constructor
+loadOutput(ArrayList<Course> courses): void

UML DIAGRAM FOR MARKBOOK CODE (Culminating - Adrian, Yash, Kiran)

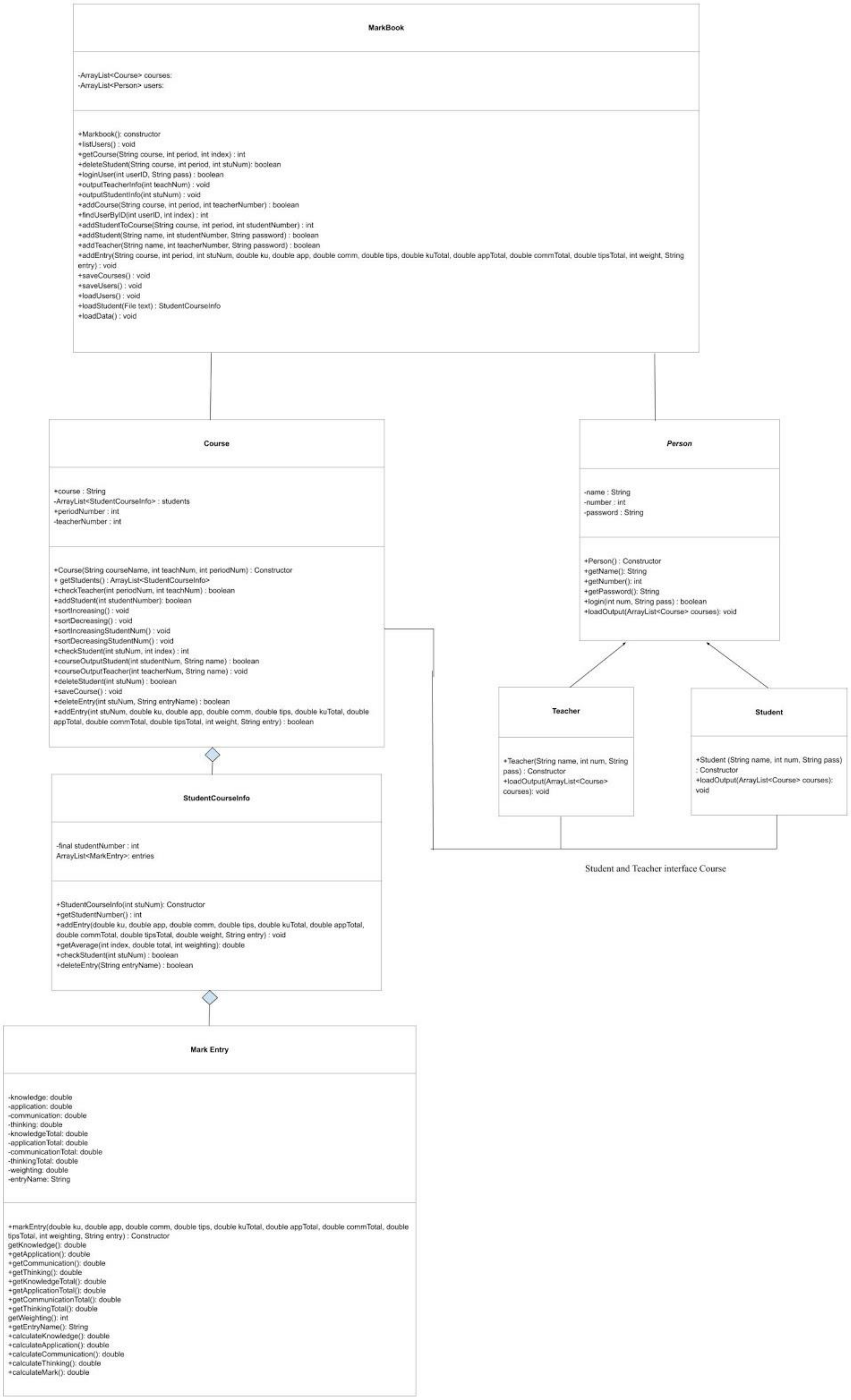
Student
+Student (String name, int num, String pass) : Constructor
+loadOutput(ArrayList<Course> courses): void

UML DIAGRAM FOR MARKBOOK CODE (Culminating - Adrian, Yash, Kiran)

MarkBook

-ArrayList<Course> courses:
-ArrayList<Person> users:

+Markbook(): constructor
+listUsers() : void
+getCourse(String course, int period, int index) : int
+deleteStudent(String course, int period, int stuNum): boolean
+loginUser(int userID, String pass) : boolean
+outputTeacherInfo(int teachNum) : void
+outputStudentInfo(int stuNum) : void
+addCourse(String course, int period, int teacherNumber) : boolean
+findUserByID(int userID, int index) : int
+addStudentToCourse(String course, int period, int studentNumber) : int
+addStudent(String name, int studentNumber, String password) : boolean
+addTeacher(String name, int teacherNumber, String password) : boolean
+addEntry(String course, int period, int stuNum, double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weight, String entry) : void
+saveCourses() : void
+saveUsers() : void
+loadUsers() : void
+loadStudent(File text) : StudentCourseInfo
+loadData() : void



```
MarkBook

-ArrayList<Course> courses:
-ArrayList<Person> users:

+Markbook(): constructor
+listUsers() : void
+getCourse(String course, int period, int index) : int
+deleteStudent(String course, int period, int stuNum): boolean
+loginUser(int userID, String pass) : boolean
+outputTeacherInfo(int teachNum) : void
+outputStudentInfo(int stuNum) : void
+addCourse(String course, int period, int teacherNumber) : boolean
+findUserByID(int userID, int index) : int
+addStudentToCourse(String course, int period, int studentNumber) : int
+addStudent(String name, int studentNumber, String password) : boolean
+addTeacher(String name, int teacherNumber, String password) : boolean
+addEntry(String course, int period, int stuNum, double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double tipsTotal, int weight, String
entry) : void
+saveCourses() : void
+saveUsers() : void
+loadUsers() : void
+loadStudent(File text) : StudentCourseInfo
+loadData() : void
```

```
Course

+course : String
-ArrayList<StudentCourseInfo> : students
+periodNumber : int
-teacherNumber : int

+Course(String courseName, int teachNum, int periodNum) : Constructor
+ getStudents() : ArrayList<StudentCourseInfo>
+checkTeacher(int periodNum, int teachNum) : boolean
+addStudent(int studentNumber): boolean
+sortIncreasing() : void
+sortDecreasing() : void
+sortIncreasingStudentNum() : void
+sortDecreasingStudentNum() : void
+checkStudent(int stuNum, int index) : int
+courseOutputStudent(int studentNum, String name) : boolean
+courseOutputTeacher(int teacherNum, String name) : void
+deleteStudent(int stuNum) : boolean
+saveCourse() : void
+deleteEntry(int stuNum, String entryName) : boolean
+addEntry(int stuNum, double ku, double app, double comm, double tips, double kuTotal, double
appTotal, double commTotal, double tipsTotal, int weight, String entry) : boolean
```

```
Person

-name : String
-number : int
-password : String

+Person() : Constructor
+getName(): String
+getNumber(): int
+getPassword(): String
+login(int num, String pass) : boolean
+loadOutput(ArrayList<Course> courses): void
```

```
Teacher

+Teacher(String name, int num, String
pass) : Constructor
+loadOutput(ArrayList<Course>
courses): void
```

```
Student

+Student (String name, int num, String pass)
: Constructor
+loadOutput(ArrayList<Course> courses):
void
```

Student and Teacher interface Course

```
StudentCourseInfo

-final studentNumber : int
ArrayList<MarkEntry>: entries

+StudentCourseInfo(int stuNum): Constructor
+getStudentNumber() : int
+addEntry(double ku, double app, double comm, double tips, double kuTotal, double appTotal,
double commTotal, double tipsTotal, double weight, String entry) : void
+getAverage(int index, double total, int weighting): double
+checkStudent(int stuNum) : boolean
+deleteEntry(String entryName) : boolean
```

```
Mark Entry

-knowledge: double
-application: double
-communication: double
-thinking: double
-knowledgeTotal: double
-applicationTotal: double
-communicationTotal: double
-thinkingTotal: double
-weighting: double
-entryName: String

+markEntry(double ku, double app, double comm, double tips, double kuTotal, double appTotal, double commTotal, double
tipsTotal, int weighting, String entry) : Constructor
getKnowledge(): double
+getApplication(): double
+getCommunication(): double
+getThinking(): double
+getKnowledgeTotal(): double
+getApplicationTotal(): double
+getCommunicationTotal(): double
+getThinkingTotal(): double
getWeighting(): int
+getEntryName(): String
+calculateKnowledge(): double
+calculateApplication(): double
+calculateCommunication(): double
+calculateThinking(): double
+calculateMark(): double
```

Stage 4: Project Review

4A. System Testing

System Testing Table:

Functionality	Purpose	Input	Expected Output	Output
General	Avoid error of invalid student number input/	Non-numeric entry or null entry.	Error message.	Error message.
General	Courses are saved.	Change is made to files.	Files saved.	Files saved.
General	Markbook loads.	Txt files.	Information loaded.	Information loaded.
Sign Up	Avoid null fields.	No-input	Error message.	Error message.
Sign Up	User already exists.	Credentials of existing user.	Error message.	Error message.
Sign Up	Password does not match confirmed password.	icsiscool vs. isciscool	Error message.	Error message.
Sign Up	User is successfully created.	Valid credentials of a new user.	Redirection to login page and addition to database.	Redirection to login page and addition to database.
Login	Invalid credentials are entered.	Invalid credentials.	Error message.	Error message.
Login	Valid credentials are entered of a student.	Valid student credentials.	Redirection to student dashboard.	Redirection to student dashboard.
Login	Valid credentials are entered of a teacher.	Valid teacher credentials.	Redirection to teacher dashboard.	Redirection to teacher dashboard.
Login	Student logs in without any courses.	Student without courses.	Error message.	Error message.
Student dashboard	Student logs in.	Student successfully logs in.	Displays student information from a course.	Displays student information from a course.
Teacher dashboard	When a teacher has no courses in the database, they can add a new course.	Teacher login without any existing course.	Query user for new course information and load information.	Query user for new course information and load information.

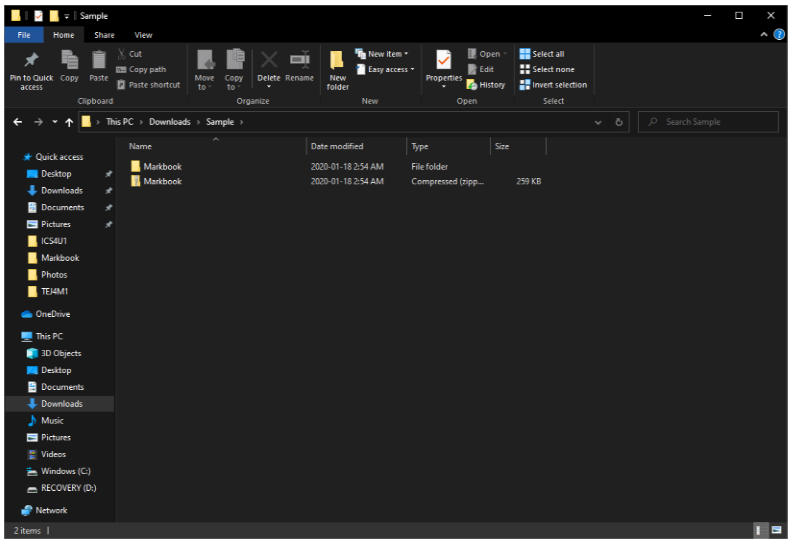
Teacher dashboard	When a student is added, mark should be 0 until a new entry added.	New student addition.	0 average.	0 average.
Teacher dashboard	When a new student is added, the student number cannot be the number of an existing student or a teacher.	Number of existing student or teacher.	Error message.	Error message.
Teacher dashboard	Student is successfully added.	Valid student number.	Student is added.	Student is added.
Teacher dashboard	When a student is deleted, the student must exist.	Number of a non-existent student.	Error message.	Error message.
Teacher dashboard	When a student is deleted, the student file must be deleted from the database.	Deletion of student.	File deleted.	File deleted.
Teacher dashboard	When a course is added, the course cannot match an existing course.	Information of existing course.	Error message.	Error message.
Teacher dashboard	When a course is added, the course period must be an integer.	Non-numeric entry.	Error message.	Error message.
Teacher dashboard	Course is added.	Valid course information.	New dashboard and course added.	New dashboard and course added.
Teacher dashboard	When a student is searched, the student must exist.	Student number of a non-existent student.	Error message.	Error message.
Teacher dashboard	When a student is searched successfully.	Student number of an existing student.	Window with student information.	Window with student information.
Teacher dashboard	Table is sorted.	Change in sorting method.	Table sorted.	Table sorted.
Teacher dashboard	When an entry is made, the entered values must be numeric.	Non-numeric entries.	Error message.	Error message.
Teacher dashboard	When an entry is made, the entry name must not match existing entries.	Entry with name matching existing entry.	Error message.	Error message.

Teacher dashboard	When an entry is made, the earned marks cannot be greater than the total marks.	Marks earned higher than mark total.	Error message.	Error message.
Teacher dashboard	An entry is successfully made.	Valid entry information.	Entry added.	Entry added.
Teacher dashboard	Entry is not deleted.	Invalid entry deletion.	Error message.	Error message.

4B. User Documentation

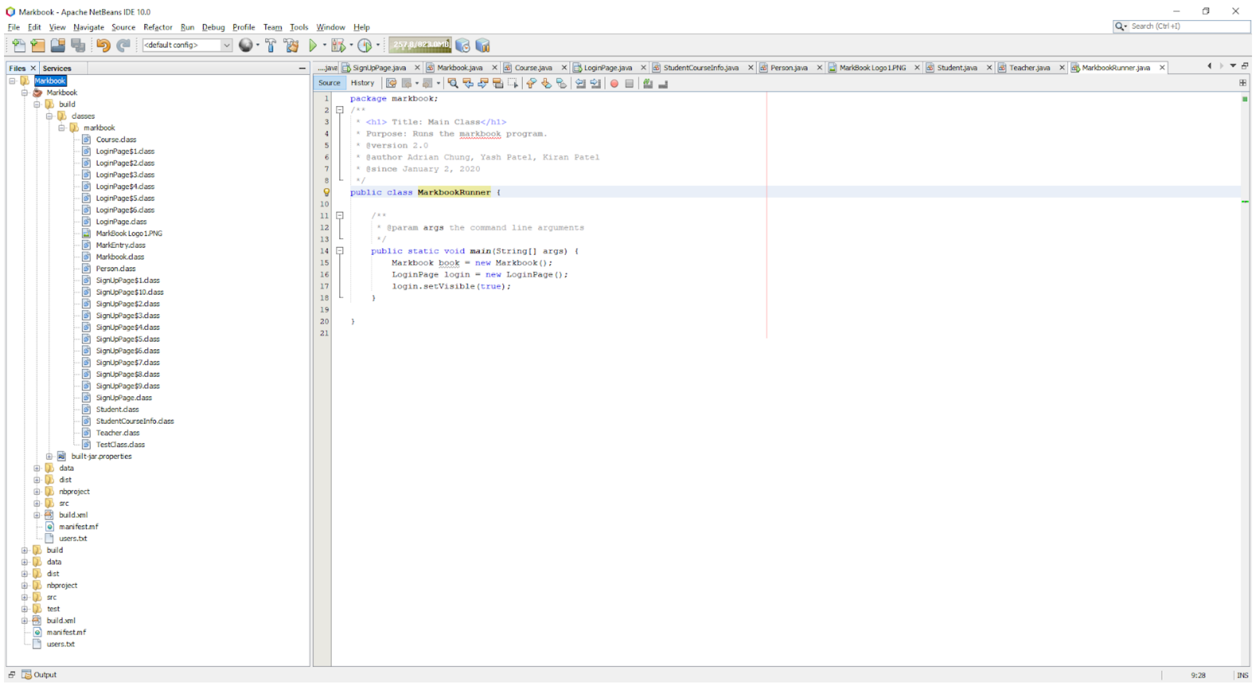
Installation:

To install, download the zip file and extract the file from the zip folder.



Name	Date modified	Type	Size
Markbook	2020-01-18 2:54 AM	File folder	
Markbook	2020-01-18 2:54 AM	Compressed (zipp...	259 KB

Then, open an appropropriate program that supports Java files, preferably Netbeans 11.0 with JDK 13. Load Markbook into the program. Select "MarkbookRunner.java", then run the file.



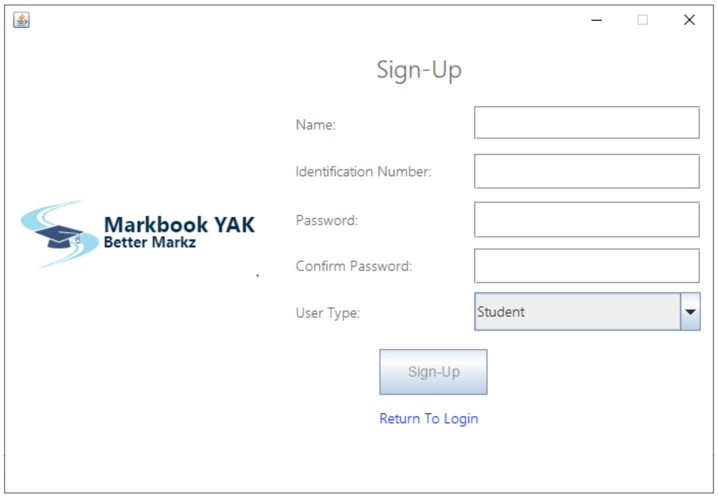
```java
package markbook;
/**
 * <h1> Title: Main Class</h1>
 * Purpose: Runs the markbook program.
 * @version 2.0
 * @author Adrian Chung, Yash Patel, Kiran Patel
 * @since January 2, 2020
 */
public class MarkbookRunner {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Markbook book = new Markbook();
        LoginPage login = new LoginPage();
        login.setVisible(true);
    }

}
```

Student Documentation:

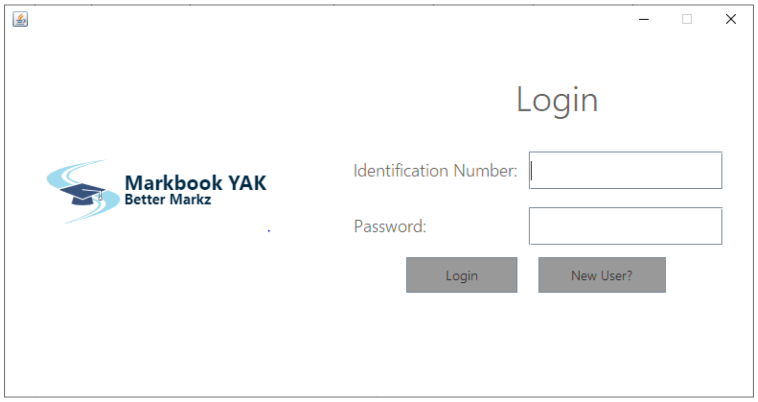
If you are a new user, please create a new account.



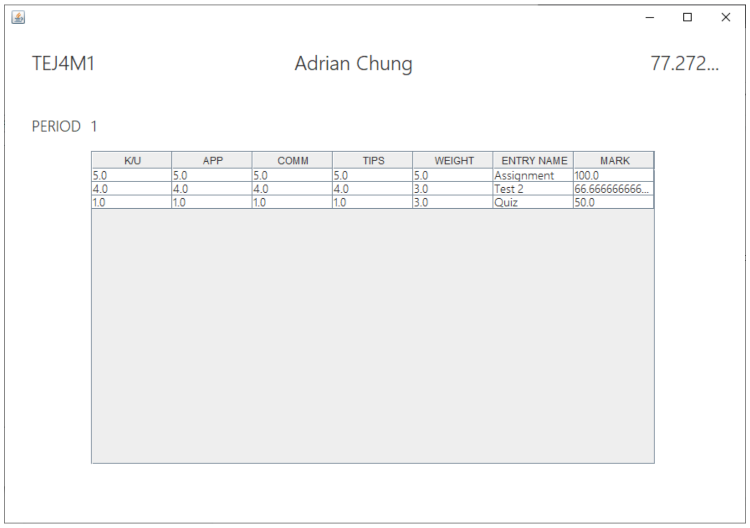
Sign-Up

Name:	
Identification Number:	
Password:	
Confirm Password:	
User Type:	Student

Fill all fields appropriately, ensuring to select student when entering student type. Upon successful sign up, you will be automatically redirected to the login page.



Upon redirection, input the information used to create your account.



TEJ4M1 Adrian Chung 77.272...

PERIOD 1

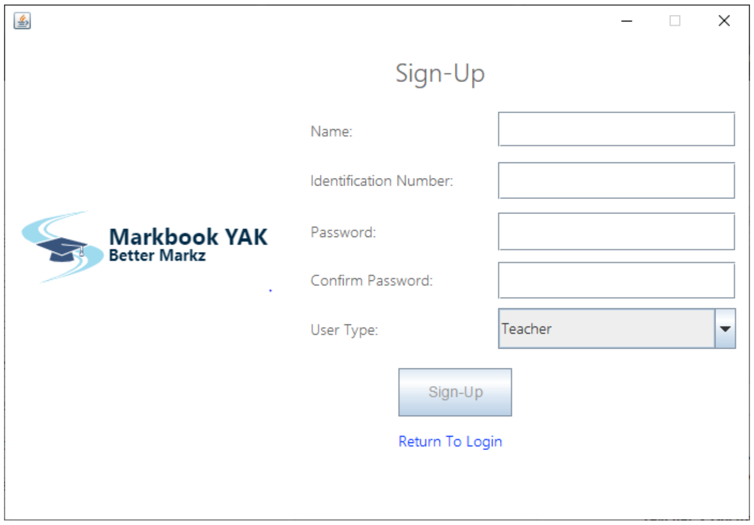
K/U	APP	COMM	TIPS	WEIGHT	ENTRY NAME	MARK
5.0	5.0	5.0	5.0	5.0	Assignment	100.0
4.0	4.0	4.0	4.0	3.0	Test 2	66.666666666...
1.0	1.0	1.0	1.0	3.0	Quiz	50.0

A new window will be created for each course you are currently enrolled in. Only courses you are enrolled in will be displayed.

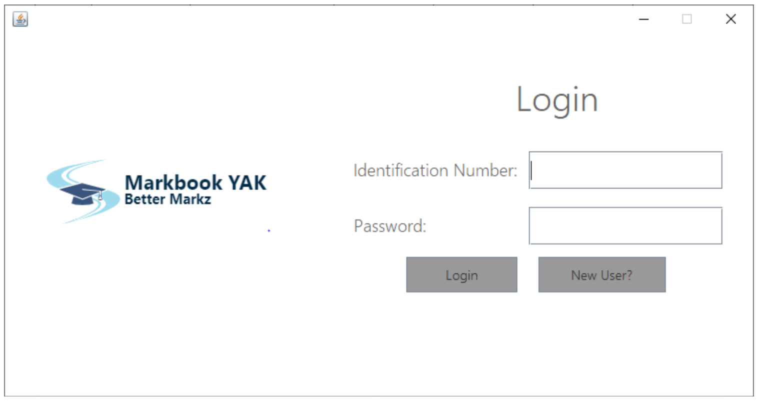
Information pertaining to the course such as period number, course name, your name, and your average will be displayed.

To log out, close the window and all windows will be close and the program will exit.

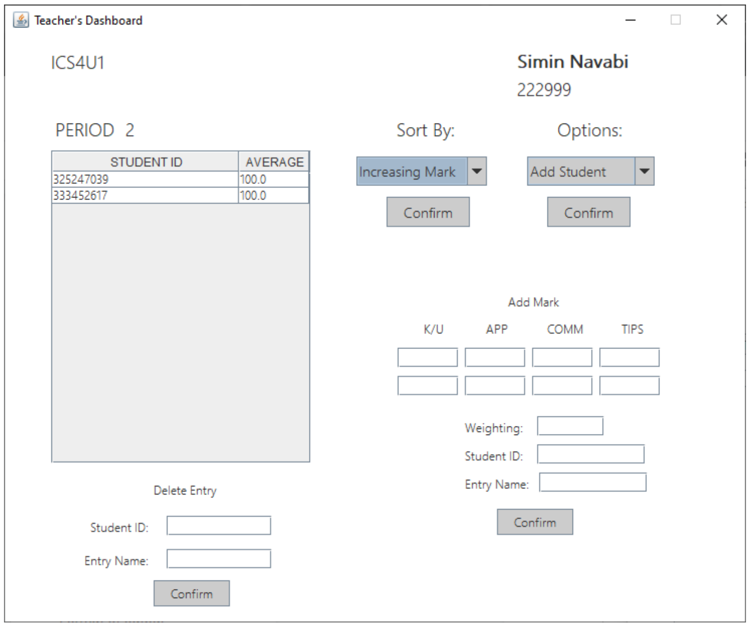
Teacher's Documentation:
If you are a new user, please create a new account.



Fill all fields appropriately, ensuring to select student when entering student type. Upon successful sign up, you will be automatically redirected to the login page.



Upon redirection, input the information used to create your account. Log in to your account and you will be redirected to your dashboard.



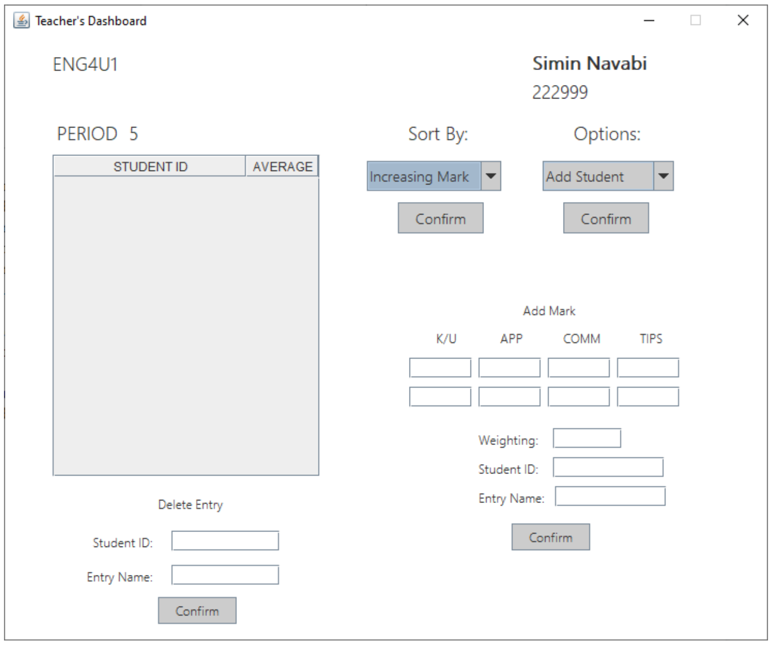
STUDENT ID	AVERAGE
325247039	100.0
333452617	100.0

Windows will be created, each representing a course you are teaching. Course information including course name, course period, your identification number, and your name will be displayed.

Within the table, you can see student information, including student number and average. In addition, there are options to sort in terms of various sorting methods. To sort, select an option from the “Sort By” box, select your choice, and confirm your selection.

Options for your action include adding a new student and deleting a student from the course. Go to the “Options” box, select the appropriate action, and confirm your selection. These actions will be immediately displayed on your window.

In addition, you may opt to create a new course. Go to the “Options” box, select “Add Course” and confirm your selection. Upon entering appropriate fields, a new window will be created.



ENG4U1

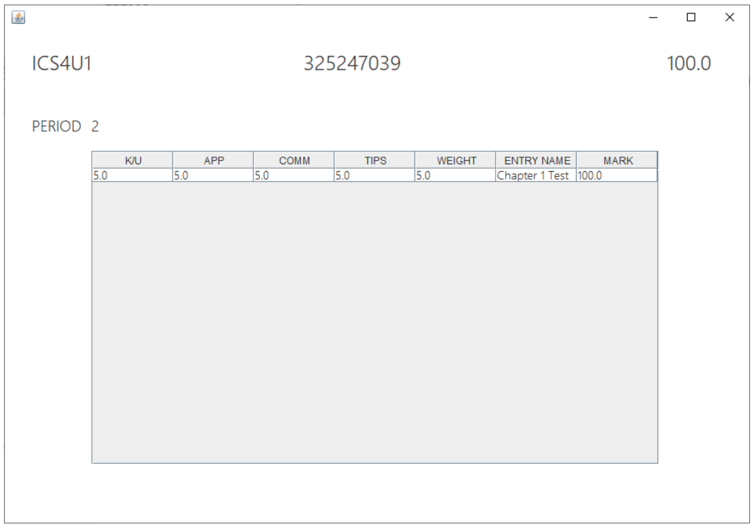
Simin Navabi
222999

PERIOD 5

Sort By: Increasing Mark

Options: Add Student

You can opt to open up a window corresponding to more detailed information regarding a particular student. Go to the “Options” box, select the “Find Student” option, and confirm your selection.



ICS4U1 325247039 100.0

PERIOD 2

K/U	APP	COMM	TIPS	WEIGHT	ENTRY NAME	MARK
5.0	5.0	5.0	5.0	5.0	Chapter 1 Test	100.0

In addition, you may add a new entry. Fill all fields under the “Add Mark” section appropriately and submit the entry. The window will be reloaded, displaying the change of the average under the appropriate student.

In addition, you may delete an entry. Fill all fields under the “Delete Entry” section appropriately and submit the entry. The window will be reloaded, displaying the change of the average under the appropriate student.

To log out, close one of the windows and the program will close.

4C: Project Review

In review of the project, the overall project was a success. The program worked with no errors, as we accounted for all possible input errors and thoroughly tested the program. All the features planned were successfully incorporated.

The program has met goals and objectives, as we have made a markbook system that both students and teachers can access. The program offers the ability for students to view their course information much more intensively and dynamically.

In addition, the program works for only valid data sets. For example, entries were only valid if they contained numerical values. In addition, the data was stored in a method that allowed for easy reading and writing of the markbook information.

Current limitations of the program includes the lack of a dynamic GUI and more flexible marking. For example, the GUI, while displaying windows for each course, would preferably display the course information in one window, with an option to switch between different courses in the window. In addition, the addition of more customizable mark entries. For example, some assessments mark only knowledge and application categories.

Additional features that could have been implemented is the ability for teacher feedback, interval sorting, sorting by student number, etc. However, that would likely consist of a GUI redesign, as the current GUIs are not suited to store that type of information. For example, to show interval sorting, there would likely need to be a new scroll panel implemented in the Teacher Dashboard. In addition, additional student information categories may be implemented, such as absences, lates, learning skills, etc.

We met all the deadlines, as we planned out the project accordingly and worked as a team in a command-and-conquer method to develop the program.

The work for the backend was considerably difficult, so having another member working on the backend of the program would have helped speed up the process of development. However, having members work on separate aspects of the project allowed for things to be implemented in a timely manner. A difficulty experienced was also the inability for multiple people to work on the program, which lead to the division of the aspects into frontend and backend.

The project components could have been managed much more thoroughly through the implementation of a prototype program and expanding the program to encompass the full breadth of the program via the addition of features. This could have allowed for a more clear program. The development of the program included the attempt to implement all the features from the start. This was more difficult, as it caused more bugs and issues that had to be resolved.

Daily Log

Date	Adrian Chung	Yash Patel	Kiran Patel
Dec 16	Discuss ideas for project.	Discuss ideas for project.	Discuss ideas for project

Dec 17	Determine project functions, begin project analysis.	Designed prototype diagrams.	Writing up prototype solutions.
Dec 18	Test certain algorithms to be used, begin developing data structure.	Designed prototype diagrams.	Began developing data structures
Dec 19	Create ideas for algorithms.	Researched data structures.	Designed prototype GUI
Dec 20	Develop mark entry and student course information classes.	Researched program ideas and methods.	Solved issues in mark entry.
Winter Break	Developed course class.	Researched program ideas and methods.	Designed Login GUI.
Jan 6	Developed file saving and data structure for course.	Designed Student Dashboard GUI.	Designed Login GUI.
Jan 7	Developed markbook, person, and student class.	Designed Student Dashboard GUI.	Designed Sign-Up GUI.
Jan 8	Developed markbook, person, and student class.	Designed Teacher Dashboard GUI.	Designed Sign-up GUI.
Jan 9	Developed file loading and data structure for data.	Designed Teacher Dashboard GUI.	Assisted with Teacher Dashboard GUI
Jan 10	Student GUI implementation.	Created GUI features.	Created GUI features.
Weekend	Login and sign up GUI implementation	Created GUI features.	Created GUI features.
Jan 13	Teacher GUI implementation.	GUI touchups.	GUI touchups.
Jan 14	Teacher GUI implementation.	Bug testing.	Bug testing.
Jan 15	Bug testing.	Bug testing.	Bug testing.
Jan 16	Bug testing.	Bug testing.	Bug testing.
Jan 17	Bug testing.	Bug testing.	Bug testing.
Weekend	GUI altering and project review.	Project review.	Edited UML diagram