**Architechtural Design**

# Classes and public methods

## [class Main](#_class_Main)

* + main(String[] args)

## [class UI](#_class_UI)

* + UI()
  + runPromptID(): void
  + promptID(): boolean
  + runPromptCommand(): void
  + promptCommand(): boolean
  + showWelcomeMsg(int): void
  + showFinishMsg(): void
  + checkID(int): boolean

## [class GradeSystems](#_class_GradeSystems)

* + GradeSystems()
  + showGrade(int): int[]
  + showRank(int): int
  + showAvg(): int
  + updateWeights(float[]): void
  + containsID(int): boolean
  + getName(int): String

## [class Grades](#_class_Grades)

* + Grades(String, int, int[], float[])
  + calculateTotalGrade(float[]): int
  + getID(): int
  + getName(): String
  + getScores(): int[]
  + getTotalGrade(): int
  + resetTotalGrade(float[]): int

# Detailed design

## class Main

/\*\* main(String[] args)

\* TIME COMPLEXITY: N/A (main)

\* PURPOSE: Initializes the system by creating a new UI() instance.

\* **@param** args

\* **@exception** IOException

\*/

public static void main(String[] args) throws IOException {

construct UI()

}

## class UI

/\*\* class UI

\* CLASS PURPOSE: UI class handles user input and calls the appropriate

\* methods from the GradeSystem class according to the commands input by

\* the user.

\*\*/

GradeSystems **aGradeSystem**;

int **userID**; // current user ID

String **userName**; // current user name

/\*\* UI()

\* TIME COMPLEXITY: N/A (UI)

\* PURPOSE: Class Constructor. Runs the runPromptID() function.

\* **@exception** IOException

\*/

public UI() throws IOException {

call GradeSystems() to construct **aGradeSystems**

call runPromptID()

}

/\*\* runPromptID()

\* TIME COMPLEXITY: N/A (UI)

\* PURPOSE: Wrapper function for the promptID() function. Prompts the user

\* to enter ID and handles user input in this state of the system of waiting for ID to

\* query.

\* **@exception** IOException

\*/

public void runPromptID() throws IOException {

try {

while promptID() returns true

call runPromptCommand()

} catch (NoSuchIDExceptions) {

show no such ID message

call runPromptID() again

}

}

/\*\* promptID()

\* TIME COMPLEXITY: N/A (UI)

\* PURPOSE: Function for prompting user to input an ID and handling user

\* input during this state of the system of waiting for ID to query. Does the

\* following:

\* [1] Prints the message "Enter your ID or 'Q' to quit:"

\* [2] If the line is the letter "Q", it quits the system by calling the

\* showWelcomeMessage() function

\* [3] Otherwise we check if the line is a valid line, if not, we throw a

\* NoSuchIDException

\* [4] If the ID is valid, we show the welcome message by calling the

\* showWelcomeMsg(ID) message

\* **@exception** NoSuchIDExceptions, IOException

\*/

public boolean promptID() throws NoSuchIDExceptions, IOException {

print "Enter your ID or 'Q' to quit:"

if user enters ‘Q’ {

call showFinishMsg()

return false

} else user enters ID{

if checkID(ID) returns false{

throw NoSuchIDExceptions

} else {

set **userID** as ID

call showWelcomeMsg(ID)

return true

}

}

}

/\*\* runPromptCommand()

\* TIME COMPLEXITY: N/A (UI)

\* PURPOSE: Wrapper function for the promptCommand() function. Prompts

\* the user to enter a command after entering an ID and handles user input in this

\* state of the system of querying a student ID.

\* @exception IOException

\*/

public void runPromptCommand() throws IOException {

try {

keep calling promptCommand() until false returned

} catch (NoSuchCommandExceptions) {

show no such command message

call runPromptCommand() again

}

}

/\*\* promptCommand()

\* TIME COMPLEXITY: N/A (UI)

\* PURPOSE: Function for prompting user to input a command after inputing an

\* ID and handling user input during this stage of querying ID. Does the following:

\* [1] Prints the list of possible commands.

\* [2] Decodes the commands, by user input:

\* ["G"] Handles the Grade command, prints the grades for each of the 5

\* exams and the final grade of the student. Calls the showGrade(ID)

\* method from the GradeSystem class.

\* ["R"] Handles the Rank command, prints the rank of the student in the

\* GradeSystem. Calls the showRank(ID) method from the

\* GradeSystem class.

\* ["A"] Handles the Average command, prints the average for the scores

\* of all students for each one of the 5 exams and the total average

\* score. Calls the showAvg() method from the GradeSystem class.

\* ["W"] Handles the Weights command.Does the following:

\* [a] Prints the list of the weights currently in the system for each

\* exam.

\* [b] Prompts user to input a new weight for each exam

\* [c] Prompts the user to reenter the new weights for each exam

\* [d] Prompts the user to confirm or deny the change ("Y" or "N")

\* [e] Changes the weights in the GradeSystem by calling the

\* method updateWeights() with the new weights list as

\* arguments

\* ["E"] Handles the Exit command. Exits the ID querying state and

\* returns to the Waiting input for ID to query state.

\* ["DEFAULT"] Any other command, throw a

\* NoSuchCommandException

\* @exception NoSuchCommandExceptions, IOException

\*/

public boolean promptCommand() throws NoSuchCommandExceptions, IOException {

print command list

case "G": // show grades of the user

call **aGradeSystem**.showGrade(**userID**) to get grades for userID

print grades

case "R": // show rank of the user

call **aGradeSystem**.showRank(**userID**) to get rank for userID

print rank

case "A": // show average grades of the class

call **aGradeSystem**.showAvg() to get class average grades

print class average grades

case "W": // update weights

print current weights

print "Enter new weights (in percentage):"

user enters new weights

print new weights for user to comfirm

print "Save? (Y/N): "

if user enters ‘Y’

call **aGradeSystem**.updateWeights(newWeights)

case "E": // exit

return false

default: // invalid command

throw NoSuchCommandExceptions

return true;

}

/\*\* showWelcomeMsg(int ID)

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Gets the name of the user identified with the most recently

\* inputted ID and outputs the "Welcome username!" message.

\* @param ID int ID of the student whose records we want to query

\*/

public void showWelcomeMsg(int ID) {

call **aGradeSystem**.getName(ID) to set **userName**

print "Welcome userName !"

}

/\*\* showFinishMsg()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Prints the goodbye message "Goodbye".

\*/

public void showFinishMsg() {

print "Goodbye"

}

/\*\* checkID(int ID)

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Checks whether an ID is in the GradeSystem. Calls the

\* containsID from the aGradeSystem GradeSystem instance.

\* @return True/False boolean true if the ID is in sList, false otherwise

\*/

public boolean checkID(int ID) {

call **aGradeSystem**.containsID(ID) to check if ID is in **aGradeSystem**

}

## class GradeSystems

/\*\* class GradeSystems

\* CLASS PURPOSE: The GradeSystems defines a System that keeps records for

\* each students of their grades and their final scores, as well as

\* keeping the average score for the entire class.

\* DATA STRUCTURES: We used a linked list of instances of the Grades class,

\* one per student.

\*/

float[] **weights** = { 0.1 ,0.1 ,0.1 ,0.3 ,0.4 }; // default weights

LinkedList<Grades> **aList**; // store the grades for the whole class

Grades **classAvg**; // grades of the class average

/\*\* GradeSystems()

\* TIME COMPLEXITY: O(n^2) worst case

\* PURPOSE: Class constructor. Builds the Grades Systems by:

\* [1] Reading grades from input file "gradeinput.txt"

\* [2] Creating a Grades instance for each student/ID .

\* [3] Calculates the class Average.

\* NOTE: Supports UTF-8 Characters.

\* @throws IOException

\*/

GradeSystems() throws IOException {

read in file “gradeinput.txt”

while (!EOF) {

call buildGrade(line) to build **aGrade** by one line from the input file

call orderedInsert(**aGrade**) to orderly insert **aGrade** into **aList**

}

call calculateClassAvg() to get **classAvg** grades

}

/\*\* showGrade(int ID)

\* TIME COMPLEXITY: O(n) worst case

\* PURPOSE: Gets the scores for each one of the 5 exams and the final grade

\* of the student identified with the ID.

\* @param ID int ID of the student whose grade we want

\* @return g.getScores() int[5] Array of scores for ID

\*/

public int[] showGrade(int ID) {

for each Grades **g** in **aList** {

if **g**.getID() equals to inputted ID

return g.getScores();

}

}

/\*\* showRank(int ID)

\* TIME COMPLEXITY: O(n) worst case

\* PURPOSE: Gets the rank of a student in the Grade System.

\* @param ID int ID of the student whose rank we want

\* @return rank int Shows the rank of the student, if other students have same

\* final grade, all of them share same rank

\*/

public int showRank(int ID) {

linearly search to find the **index** (in **aList**) and **totalGrade** of the inputted ID

keep calling Grades **g** = **aList**.get(--**index**) until **g**.totalGrade != **totalGrade**

**index**++; // index = min{ i | aList[i].totalGrade == **totalGrade**}

return **index**+1; // index 0 is rank 1, index 1 is rank 2, etc.

}

/\*\* showAvg()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Gets the array of 6 average grades for the students in the system

\* for each exam ant for the final grade.

\* @return classAvg.getScores() int[5] Representing the grades for each exam

\*/

public int[] showAvg() {

call classAvg.getScores()

}

/\*\* updateWeights(float[] weights)

\* TIME COMPLEXITY: O(n^2)

\* PURPOSE: Recalculates all grades for students based on new weights.

\* Iterates through the aList of Grade instances and calls the

\* resetTotalGrade(weights) for each of them.

\* @param weights float[5] Representing the grades for each exam

\*/

public void updateWeights(float[] weights) {

this.weights = weights

for each Grades **g** in aList

call **g**.resetTotalGrade(weights)

copy aList to a temporary List **tmp**

aList.clear();

for each Grades **g** in **tmp**

call orderedInsert(**g**)

}

/\*\* containsID(int ID)

\* TIME COMPLEXITY: O(n) worst case

\* PURPOSE: Checks whether an ID is in the GradeSystem. Iterates linearly

\* through the aListand checks their ID.

\* @param ID int ID of the student

\* @return True/False boolean true if the ID is in sList, false otherwise

\*/

public boolean containsID(int ID) {

for each Grades **g** in aList

if **g**.getID() equals to inputted ID

return true;

}

/\*\* getName(int ID)

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Gets the name of a student ID in the system.

\* @param ID Int ID of the student

\* @return ID.getName() String The string corresponding to the ID's name

\*/

public String getName(int ID) {

for each Grades **g** in aList

if **g**.getID() equals to inputted ID

return **g**.getName();

}

## class Grades

/\*\* class Grades

\* CLASS PURPOSE: Grades class stores the information for each tuple of the

\* database for the Grading System.

\* Its attributes are ID, name and the grades for each one of the five exams:lab1,

\* lab2, lab3, midTerm, finalExam, and totalGrade.

\*\*/

int ID; // student ID

String name; // student name

int[] exams; // grades of lab1, lab2, lab3, midTerm, finalExam and totalGrade

/\*\* Grades()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Constructor, assigns values to each attribute of the instance. Calls

\* CalculateTotalGrade() to assign a value to the totalGrade attribute, takes

\* arguments for the rest of the arguments.

\* @param name String representing name for student

\* @param id int representing the ID of the student, id 0 is reserved for the

\* class average

\* @param lab1,lab2,lab3,midTerm,finalExam int corresponding to the grades

\* for those exams

\* @param weights float[5] array with the weight for each exam

\*\*/

public Grades(String name, int ID, int[] scores, float[] weights) {

this.ID = ID;

this.name = name;

for i = 0~4 exams[i] = scores[i];

exams[5] = calculateTotalGrade(weights);

}

/\*\* calculateTotalGrade(float[] weights)

\* TIME COMPLEXITY: O(n)

\* PURPOSE: Calculates the total grade as a weighted average of the 5 grades in

\* the attributes for the ID. The weighted average is rounded to the closest integer.

\* @param weights float[4] array for the weights of each exam

\* @return total int representing rounded weighted average of the 5 grades

\*/

public int calculateTotalGrade(float[] weights) {

for i = 0~4

total += exams[i] \* weights[i]

return round(total)

}

/\*\* getID()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Getter method for the student's ID.

\* @return this.ID int representing student's ID

\*/

public int getID() {

return this.ID;

}

/\*\* getName()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Getter method for the student's name.

\* @return this.name string representing the student's name

\*/

public String getName() {

return this.name;

}

/\*\* getScores()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Getter method for the scores.

\* @return this.exams int[5] representing the grades for each exam

\*/

public int[] getScores() {

return this.exams;

}

/\*\* getTotalGrade()

\* TIME COMPLEXITY: O(1)

\* PURPOSE: Getter method for the student's total grade.

\* @return this.exams[5] int representing the total averaged Grade

\*/

public int getTotalGrade() {

return this.exams[5];

}

/\*\* resetTotalGrade(float[] weights)

\* TIME COMPLEXITY: O(n)

\* PURPOSE: Recalculates the Student's Total average grade with the weights

\* provided.

\* @param weights float[4] array for the weights of each exam

\* @return this.exams[5] int representing the total averaged Grade

\*/

public int resetTotalGrade(float[] weights) {

this.exams[5] = calculateTotalGrade(weights);

return this.exams[5];

}