Goals & Objectives

The goal of this program is to read in data from a file titled student.txt, calculating the average Grade Point Average (GPA), and outputting to the user those students that meet the requirements for membership in the Honor Society.

Lessons Learned

Overall, this was a fun project, learning how to get multiple classes, including super and sub classes to work together. The two biggest lessons I learned through this project were 1) calling super class methods and variables from subclasses and 2) being able to store all objects created in one ArrayList vs having multiple ArrayLists. These lessons played in to each other as I worked on the project.

1. Since Undergraduate and Graduate are sub classes to Student, I found I was able to use the call super() in the constructor which created a Student object, as well as the Undergraduate/Graduate object to match. This allowed for me to call the getGPA() method from Student, but use Undergraduate or Graduate’s eligibleForHonorSociety() methods.
2. At first, I began with 2 ArrayLists, one for Graduate and one for Undergraduate, and iterated through each student in the file and added them to their corresponding ArrayList. This worked until I needed to sum all GPAs, which complicated the code. I then switch to a single Student object ArrayList, and stored all students in it, but calling the corresponding buildX() method to build the Undergraduate or Graduate object. This allowed me to easily sum all GPAs, and find those eligible for Honor Society, since each subclass had that method, which called the super class method to check the GPA, and the subclass method to check any additional requirements.

Functional Requirements

1. Read students.txt and store information.
2. Determine Student status as Graduate or Undergraduate.
3. Calculate the GPA of each student.
4. Set GPA Threshold as the midpoint between the average GPA and 4.0
5. Output the GPA Threshold
6. Check for and output the Name, GPA, and Degree Type or School Year of each eligible Honor Society student(s).

Pseudocode

Import Java Utilities

Import Java IO

Class Project2 {

Function buildUndergraduate {

Return new Undergraduate object.

End

Function buidGraduate {

Return new Graduate object.

End

Function Main {

Declare File studentFile as a new File

Declare Student ArrayList students as a new ArrayList

IF (studentFile exists) {

Try declaring Scanner reader as a new Scanner for studentFile {

While (reader.hasNext) {

Declare String Array tmp as areader.nextLine().split(“ “)

IF (tmp[3] is “Masters” or “Doctorate”) {

Call buildGraduate and add to students

ELSE

Call buildUndergraduate and add to student

End

End

Declare double sumOfAllGPAs as 0.0

Declare int numberOfGPAs as 0

FOR (each student in students) {

Call getGPA() and add to sumOfAllGPAs

Increment numberOfGPAs

End

Declare midpointGPA as result of Student.setGpaThreshold

Output “GPA threshold for membership is: “ + midpointGPA

Output blank line

Output “Student(s) eligible for Honor Society:”

For (int i = 0; i <= students.size() -1; i++) {

IF (students.get(i).eligibleForHonorSociety == 1 {

Output students.get(i).toString()

End

End

Catch FileNotFoundException {

getLocalizedMessage()

ELSE

Output “File Not Found”

Throw new FileNotFoundException

End

End

End

Class Student {

Declare String studentName as private

Declare int creditHours as private

Declare int qualityPoints as private

Declare double gpa as private

Declare double gpaThreshold as private and static

Constructor Student {}

Constructor Student (requires String name, int creditHours, and int qualityPoints) {

Assign this.studentName as name

Assign this.creditHours as creditHours

Assign this.qualityPoints as qualityPoints

Assign this.gpa as calculateGPA result

End

Function calculateGPA {

Return Math.round(qualityPOints / creditHours) as double

End

Function getGPA {

Return this.gpa

End

Function eligibleForHonorSociety {

IF (this.gpa >= gpaThreshold) {

Return 1

ELSE

Return 0

End

End

Static Function setGpaThreshold {

Declare averageGPA as Math.round(sumOfAllGPAs / numberOfGPAs) as double

Assign gpaThreshold as Math.round((averageGPA \* 4.0) / 2

Return gpaThreshold

End

End

Class Undergraduate extends Student {

Declare String schoolYear as private and final

Constructor Undergraduate (requires String name, int creditHours, int qualityPoints, String

schoolYear) {

Call superclass

Assign this.schoolYear as schoolYear

End

Overriding Function eligibleForHonorSociety {

IF (this.schoolYear is “Junior” or “Senior” AND super.eligibleForHonorSociety equals 1) {

Return 1

ELSE

Return 0

End

End

Overriding Function toString {

Return super.toString() + “ “ + this.schoolYear.toUpperCase()

End

End

Class Graduate extends Student {

Declare String degreeType as private and final

Constructor Graduate (requires String name, int creditHours, int qualityPoints, String

degreeType) {

Call superclass

Assign this.degreeType as degreeType

End

Overriding Function eligibleForHonorSociety {

IF (this.schoolYear is “masters” AND super.eligibleForHonorSociety equals 1) {

Return 1

ELSE

Return 0

End

End

Overriding Function toString {

Return super.toString() + “ “ + this.degreeType.toUpperCase()

End

End

UML Diagram

|  |
| --- |
| **Project2** |
| -studentFile: File |
| -students: ArrayList |
| -sumOfAllGPAs: double |
| -numberOfGPAs: int |
| -midpointGPA: double |
| +buildUndergrad(inputs: String[]): Undergraduate  +buildGrad(inputs: String[]): Graduate |

|  |
| --- |
| **Student** |
| -studentName: String |
| -creditHours: int |
| -qualityPoints: int |
| -gpa: double |
| -gpaThreshold: double |
| +Student() |
| +Student(name: String, creditHours: int, qualityPoints: int) |
| +calculateGPA(): double |
| +getGPA(): double |
| +eligibleForHonorSociety(): int |
| +toString(): String |

|  |
| --- |
| **Undergraduate** |
| -schoolYear: String |
| +Undergraduate(name: String, creditHours: int, qualityPoints: int, schoolyear: String) |
| +eligibleForHonorSociety(): int |
| +toString(): String |

|  |
| --- |
| **Graduate** |
| -degreeType: String |
| +Graduate(name: String, creditHours: int, qualityPoints: int, degreeType: String) |
| +eligibleForHonorSociety(): int |
| +toString(): String |

Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1a | Read students.txt and store information. | Program reads data from students.txt, creates appropriate object, and stores in students ArrayList | Scanner reader = new Scanner(studentsFile);  While (reader.hasNext())  Students.add(buildGrad(tmp));  Students.add(buildUndergrad(tmp)); | Pass |
| 2a | Determine Student status as Graduate or Undergraduate. | Create temporary array and store student elements. Check if last array item is “masters” or “doctorate” | String[] tmp = reader.nextLine().split(“ “);  If(tmp[3].equalsIgnoreCase(“masters”) || tmp[3].equalsIgnoreCase(“doctorate”)) | Pass |
| 3a | Calculate the GPA of each student. | Undergraduate and Graduate constructors call superclass constructor and calculateGPA() method | Constructor passes require variables to Student superclass. Student sets gpa with method calculateGPA() which returns a rounded double from qualityPoints / creditHours. | Pass |
| 4a | Set GPA Threshold as the midpoint between the average GPA and 4.0 | Sum all GPAs, count the number of GPAs, and call method that calculates the average GPA and the midpoint GPA. | For each student in students, add student.getGPA to sumOfAllGPAs and increment numberOfGPAs. Call setGpaThreshold(), which calculates the average GPA, then gets the midpoint between the average GPA and 4.0, and sets instance variable gpaThreshold and returns gpaThreshold. | Pass |
| 5a | Output the GPA Threshold | Output “GPA Threshold for membership is: “ + gpaThreshold. | Output “GPA Threshold for membership is: “ + result of method setGpaThreshold(). | Pass |
| 6a | Check for and output the Name, GPA, and Degree Type or School Year of each eligible Honor Society student(s). | Check each students GPA against the GPA Threshold for membership eligibility, and output Name, GPA, and Degree Type or School Year if eligible. | Call each student’s eligibleForHonorSociety() method, which checks GPA, school year or degree type, and returns 1 if eligible. Output Name, GPA, school year or degree type if value was a 1. | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1 | Brown,William 72 230 Junior  Johnson,Mary 21 77 Masters  Jones,Sally 32 95 Sophomore | GPA threshold for membership is 3.64  Student(s) eligible for Honor Society:  Name: Johnson,Mary, GPA: 3.67 MASTERS | GPA threshold for membership is 3.64  Student(s) eligible for Honor Society:  Name: Johnson,Mary, GPA: 3.67 MASTERS | Pass |
| 2 | Baskin,Joe 29 106 Sophomore  Baskin,Stefani 78 298 Senior  Jones,Sally 55 198 Junior | GPA threshold for membership is 3.85  Student(s) eligible for Honor Society: | GPA threshold for membership is 3.85  Student(s) eligible for Honor Society: | Pass |
| 3 | Rash,Madison 29 95 Freshman  Devilla,Hayden 50 200 Junior  Jones,Sally 84 246 Masters | GPA threshold for membership is 3.7  Student(s) eligible for Honor Society:  Name: Devilla,Hayden GPA: 4.00 JUNIOR | GPA threshold for membership is 3.7  Student(s) eligible for Honor Society:  Name: Devilla,Hayden GPA: 4.00 JUNIOR | Pass |
| 4 | Rash,Madison 29 95 Freshman  Devilla,Hayden 50 200 Junior  Jones,Sally 84 246 Doctorate  Brown,William 72 230 Junior  Johnson,Mary 21 77 Masters  Baskin,Joe 29 106 Sophomore  Baskin,Stefani 78 298 Senior  Burgandy,Ron 55 198 Junior | GPA threshold for membership is 3.76  Student(s) eligible for Honor Society:  Name: Devilla,Hayden GPA: 4.00 JUNIOR  Name: Baskin,Stefani GPA: 3.82 SENIOR | GPA threshold for membership is 3.76  Student(s) eligible for Honor Society:  Name: Devilla,Hayden GPA: 4.00 JUNIOR  Name: Baskin,Stefani GPA: 3.82 SENIOR | Pass |

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated