1. Nick Wright CSC285 Problem 2 part 2
2. Java Code

/\* This is a program that is capable of reading in student information from an input file, and putting it in an array of student objects.  
 \* The Program will then print out student information when requested as well as use the functions AddStudent and DeleteStudent  
 \* to add and remove students from the class as needed.  
 \* The program also utilizes the method SortLarge in order to sort students in the class from highest to lowest based upon their percentage in the class.  
 \* This program contains three classes, class Student1, class Student2, and public class CodeforProb2CSC285\*/  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
/\* class Student1 contains the object Student1 constructors and variables used in Student1 objects  
 \* class Student1 implements Comparable so that the compareTo function can be overridden  
 \* class Student1 is not utilized for part 2 of the assignment \*/  
class Student1 implements Comparable{ //implementing comparable in order to override the compareTo function  
 protected int pscore; //this is the percent score for the student  
 protected int[] testScore = new int[3]; //testScore is an integer array for the test scores of a student  
 protected String id; //String id is the variable to hold the student id  
 protected String name; //String name is the variable that will hold the ame of the student  
 protected String letterGrade; //a string variable that is used to store the letter grade  
  
 public Student1(){}; //default constructor for student  
 public Student1(int[] test, String id, String name){ //this is the constructor for the student  
 this.id=id;  
 this.name=name;  
 for(int i=0; i<3; i++){ //for loop is used to put the test scores in the testScore array  
 this.testScore[i]=test[i];  
 }  
 pscore = (int)(((test[0]+test[1]+test[2])/3.0)+0.5); //calculates the average test score as an integer  
  
 if(pscore >= 90){ //if else statement chain that will calculate letterGrade based on the pscore variable  
 letterGrade = "A";  
 }else if((pscore<90) && (pscore>=80)){  
 letterGrade = "B";  
 }else if((pscore<80) && (pscore>=70)){  
 letterGrade = "C";  
 }else if((pscore<70) && (pscore>=60)){  
 letterGrade = "D";  
 }else{  
 letterGrade = "F";  
 }//end of if else statement chain  
 } //end of Student1 object constructor  
  
 public int getPscore() { //a getter method for pscore that is used in the comapreTo method  
 return pscore;  
 } //getter method for pscore  
  
 @Override  
 public String toString() { //changing the format for when we want to print out an object  
 return id+" | "+name+" | "+testScore[0]+" | "+testScore[1]+" | "+testScore[2]+" | "+pscore+"% | "+letterGrade;  
 }  
  
 @Override  
 public int compareTo(Object o) { //overriding the compareTo function  
 if(getPscore()>((Student1)o).getPscore()){  
 return 1; //returning 1 if object 1 is larger than object 2  
 } else if (getPscore()<((Student1)o).getPscore()){  
 return -1; //returning -1 if object 1 is smaller than object 2  
 } else {  
 return 0; //will return 0 if objects are the same  
 }  
 }  
} //end of Student1 class  
  
/\* class Student2 contains the object Student2 constructors and variables used in Student2 objects  
 \* class Student2 implements Comparable so that the compareTo function may be overridden  
 \* class Student2 is utilized in part 2 of the assignment \*/  
class Student2 implements Comparable{  
 protected int pscore; //this is the percent score for the student  
 protected int[] testScore = new int[3]; //testScore is an integer array for the test scores of a student  
 protected String id; //String id is the variable to hold the student id  
 protected String name; //String name is the variable that will hold the ame of the student  
 protected String letterGrade; //a string variable that is used to store the letter grade  
 protected int TotalNoHours; //variable to hold the total hours a student has taken  
 protected float CumulativeGPA; //variable to hold a student's current GPA  
 protected String studentClass;//this variable holds the year of college that a student is is in  
 protected float newStudentGPA; //this variable will hold the updated GPA of a student based upon their GP in the current class  
  
 public Student2(){} //the default constructor for Student2  
 public Student2(int[] test, String id, String name, int TotalNoHours, float CumulativeGPA){ //Student2 object constructor with parameters  
 this.id=id;  
 this.name=name;  
 for(int i=0; i<3; i++){ //for loop is used to put the test scores in the testScore array  
 this.testScore[i]=test[i];  
 }  
 this.TotalNoHours=TotalNoHours;  
 this.CumulativeGPA=CumulativeGPA;  
  
 pscore = (int)(((test[0]+test[1]+test[2])/3.0)+0.5); //calculates the average test score as an integer  
 if(pscore >= 90){ //if else statement chain that will calculate letterGrade based on the pscore variable  
 letterGrade = "A";  
 }else if((pscore<90) && (pscore>=80)){  
 letterGrade = "B";  
 }else if((pscore<80) && (pscore>=70)){  
 letterGrade = "C";  
 }else if((pscore<70) && (pscore>=60)){  
 letterGrade = "D";  
 }else{  
 letterGrade = "F";  
 }//end of if else chain  
  
 if(TotalNoHours <= 30){ //if else statement chain is used to calculate which grade a student is in depending on the amount of credit hours they have taken.  
 studentClass = "FR";  
 }else if((TotalNoHours > 30) && (TotalNoHours <= 60)){  
 studentClass = "SO";  
 }else if((TotalNoHours > 60)&&(TotalNoHours<=90)){  
 studentClass = "JR";  
 }else if(TotalNoHours>90){  
 studentClass = "SR";  
 } //end of if else chain  
  
  
 if(letterGrade.equals("A")){ //if else chain will calculate the new GPA of a student, variations in the formula occur depending on the student's letter grade in the class.  
 newStudentGPA = ((CumulativeGPA\*TotalNoHours)+(2)\*4)/(TotalNoHours+2); //a letter grade of A means that we multiply by 4  
 newStudentGPA = (int)(newStudentGPA\*100); //I am getting rid of extra decimals in the GPA, so that they have a maximum of 2 decimal places  
 newStudentGPA = newStudentGPA/100;  
 }else if(letterGrade.equals("B")){  
 newStudentGPA = ((CumulativeGPA\*TotalNoHours)+(2)\*3)/(TotalNoHours+2); //a letter grade of B means that we multiply by 3  
 newStudentGPA = (int)(newStudentGPA\*100);  
 newStudentGPA = newStudentGPA/100;  
 }else if(letterGrade.equals("C")){  
 newStudentGPA = ((CumulativeGPA\*TotalNoHours)+(2)\*2)/(TotalNoHours+2); //a letter grade of C means that we multiply by 2  
 newStudentGPA = (int)(newStudentGPA\*100);  
 newStudentGPA = newStudentGPA/100;  
 }else if(letterGrade.equals("D")){  
 newStudentGPA = ((CumulativeGPA\*TotalNoHours)+(2)\*1)/(TotalNoHours+2); //a letter grade of D means that we multiply by 1  
 newStudentGPA = (int)(newStudentGPA\*100);  
 newStudentGPA = newStudentGPA/100;  
 }else{  
 newStudentGPA = ((CumulativeGPA\*TotalNoHours)+(2)\*0)/(TotalNoHours+2); //a letter grade of F means that we multiply by 0  
 newStudentGPA = (int)(newStudentGPA\*100);  
 newStudentGPA = newStudentGPA/100;  
 }//end of if else chain  
 }//end of Student2 object constructor  
  
 public int getPscore() { //a getter method for pscore that is used in the comapreTo method  
 return pscore;  
 }// a getter method for returning the pscore variable  
  
 @Override  
 public String toString() { //changing the format for when we want to print out an object  
 return id+" | "+name+" | "+testScore[0]+" | "+testScore[1]+" | "+testScore[2]+" | "+pscore+"% | "+  
 letterGrade+" | "+TotalNoHours+" | "+CumulativeGPA+" | "+newStudentGPA+" | "+studentClass;  
 }  
  
 @Override  
 public int compareTo(Object o) { //changing compareTo function for when we use it in method SortLarge  
 if(getPscore()>((Student2)o).getPscore()){  
 return 1; //returning 1 if object 1 is larger than object 2  
 } else if (getPscore()<((Student2)o).getPscore()){  
 return -1; //returning -1 if object 1 is smaller than object 2  
 } else {  
 return 0; //will return 0 if objects are the same  
 }  
 }  
}  
  
/\* public class CodeforProb2CSC285 contains the main method  
 \* public class CodeforProb2CSC285 contains methods AddStudent, DeleteStudent, and SortLarge\*/  
public class CodeforProb2CSC285{  
 public static void AddStudent(ArrayList<Student2> Academic\_Class2, Student2 Obj){ //static method AddStudent that adds students to the array list  
 Academic\_Class2.add(Obj);  
 }  
 public static void DeleteStudent(ArrayList<Student2> Academic\_Class2, String StudentID){ //static method DeleteStudent that removes students from the array list  
 for(int i=0; i<Academic\_Class2.size(); i++){  
 if(Academic\_Class2.get(i).id.equals(StudentID)){ //comparing the id of the current student to the id of the student being removed  
 Academic\_Class2.remove(Academic\_Class2.get(i)); //if the id's match, the current student gets removed  
 }  
 }  
 }  
  
 /\* method SortLarge is utilized in order to sort students from highest to lowest based on their percentage score in the class (represented as pscore)  
 \* saveStudent and saveStudent2 are new objects that will be used to help swap objects in the array if they are out of order \*/  
 public static void SortLarge(ArrayList<Student2> Academic\_Class2){ //static method SortLarge sorts the students from largest to smallest based on their pscore variable  
 Student2 saveStudent = new Student2(); //saveStudent and saveStudent2 are new objects that will be used to help swap objects in the array  
 Student2 saveStudent2 = new Student2();  
 int whileCondition = 1; //whileCondition is an int used to end the loop  
 while(whileCondition == 1){  
 whileCondition = 0;  
 for(int i=0; i<Academic\_Class2.size()-1; i++){  
 switch (Academic\_Class2.get(i).compareTo(Academic\_Class2.get(i+1))){  
 case 1: //the objects are in the right order  
 break;  
 case -1: //the objects are out of order and must be changed  
 saveStudent=Academic\_Class2.get(i);  
 saveStudent2=Academic\_Class2.get(i+1);  
 Academic\_Class2.remove(i);  
 Academic\_Class2.add(i,saveStudent2);  
 Academic\_Class2.remove(i+1);  
 Academic\_Class2.add(i+1,saveStudent);  
 whileCondition = 1;  
 break;  
 default: //objects are equal or no change  
 }//end of switch  
 }//end of for loop  
 }//end of while loop  
 } //end of SortLarge  
  
 public static void main(String[] args) throws FileNotFoundException { //FileNotFoundException needed for reading the input file  
 ArrayList<Student2> Academic\_Class2 = new ArrayList<Student2>(); //creating the array list Academic\_Class2  
 Scanner input = new Scanner(new File("Input2.txt")); //creating the scanner to read through the input file  
  
 while(input.hasNext()){ //while loop used to iterate through the input file and assign variables their values  
 int[] test = new int[3]; //creating variables that will be used to temporarily store the values being read in from input text before they are put into an object  
 String Sid = input.next();  
 String name = input.next();  
 test[0] = input.nextInt();  
 test[1] = input.nextInt();  
 test[2] = input.nextInt();  
 int creditHour = input.nextInt();  
 float gpa = input.nextFloat();  
  
 Student2 workStu = new Student2(test, Sid, name, creditHour, gpa); //creating a new Student2 object  
 Academic\_Class2.add(workStu); //storing object in Academic\_Class2  
 }//end of while loop  
  
 System.*out*.println("List the objects from the class ArrayList including the % score and the grades.");  
 System.*out*.println("Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling");  
 for(int i =0; i < Academic\_Class2.size(); i++){ //for loop is used to iterate through the ArrayList and print out each object  
 System.*out*.println(Academic\_Class2.get(i));  
 }  
  
 *DeleteStudent*(Academic\_Class2, "42P4"); //deleting the student with the same student id  
 *DeleteStudent*(Academic\_Class2, "45A3"); //do not need to use dot syntax because the method is static  
  
 System.*out*.println("\n" + "List the ArrayList with the dropped student records.");  
 System.*out*.println("Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling");  
 for(int i =0; i < Academic\_Class2.size(); i++){ //for loop is used to iterate through the ArrayList and print out each object  
 System.*out*.println(Academic\_Class2.get(i));  
 }  
  
 *AddStudent*(Academic\_Class2, new Student2(new int[]{80,75,98},"67T4","Clouse,B", 102, (float)3.65)); //adding new students to the array list  
 *AddStudent*(Academic\_Class2, new Student2(new int[]{75,78,72},"45P5","Garrison,J", 39, (float)1.85)); //do not need to use dot syntax because method AddStudent is static  
 *AddStudent*(Academic\_Class2, new Student2(new int[]{85,95,99},"89P0","Singer,A", 130, (float)3.87));  
 *SortLarge*(Academic\_Class2); //SortLarge used to resort the class from largest to smallest letter grade, dot syntax is not needed because method SortLarge is static  
  
 System.*out*.println("\n" + "List the ArrayList after the new students have been added and their grades have been sorted from highest to lowest.");  
 System.*out*.println("Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling");  
 for(int i =0; i < Academic\_Class2.size(); i++){ //for loop is used to iterate through the ArrayList and print out each object  
 System.*out*.println(Academic\_Class2.get(i));  
 }  
 } //end of main method  
} //end of CodeforProb2CSC285

1. Input Files

45A3 Jones,H 86 88 95 98 3.42

34K5 Horner,M 67 75 23 17 1.95

56J8 Gach,T 75 85 90 60 3.75

34U8 Hunter,C 100 50 75 75 2.60

42P4 Hinrichs,S 85 75 65 52 3.29

78T6 Johnson,K 80 78 89 15 2.00

44L2 Levitte,H 56 66 99 100 2.35

88I9 Garner,J 95 92 98 110 3.89

1. Output Files

List the objects from the class ArrayList including the % score and the grades.

Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling

45A3 | Jones,H | 86 | 88 | 95 | 90% | A | 98 | 3.42 | 3.43 | SR

34K5 | Horner,M | 67 | 75 | 23 | 55% | F | 17 | 1.95 | 1.74 | FR

56J8 | Gach,T | 75 | 85 | 90 | 83% | B | 60 | 3.75 | 3.72 | SO

34U8 | Hunter,C | 100 | 50 | 75 | 75% | C | 75 | 2.6 | 2.58 | JR

42P4 | Hinrichs,S | 85 | 75 | 65 | 75% | C | 52 | 3.29 | 3.24 | SO

78T6 | Johnson,K | 80 | 78 | 89 | 82% | B | 15 | 2.0 | 2.11 | FR

44L2 | Levitte,H | 56 | 66 | 99 | 74% | C | 100 | 2.35 | 2.34 | SR

88I9 | Garner,J | 95 | 92 | 98 | 95% | A | 110 | 3.89 | 3.89 | SR

List the ArrayList with the dropped student records.

Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling

34K5 | Horner,M | 67 | 75 | 23 | 55% | F | 17 | 1.95 | 1.74 | FR

56J8 | Gach,T | 75 | 85 | 90 | 83% | B | 60 | 3.75 | 3.72 | SO

34U8 | Hunter,C | 100 | 50 | 75 | 75% | C | 75 | 2.6 | 2.58 | JR

78T6 | Johnson,K | 80 | 78 | 89 | 82% | B | 15 | 2.0 | 2.11 | FR

44L2 | Levitte,H | 56 | 66 | 99 | 74% | C | 100 | 2.35 | 2.34 | SR

88I9 | Garner,J | 95 | 92 | 98 | 95% | A | 110 | 3.89 | 3.89 | SR

List the ArrayList after the new students have been added and their grades have been sorted from highest to lowest.

Student ID | Name | Test 1 | Test 2 | Test 3 | Percent Score | Letter Grade | Credit Hours | GPA before entering class | Updated GPA | Year of Schooling

88I9 | Garner,J | 95 | 92 | 98 | 95% | A | 110 | 3.89 | 3.89 | SR

89P0 | Singer,A | 85 | 95 | 99 | 93% | A | 130 | 3.87 | 3.87 | SR

67T4 | Clouse,B | 80 | 75 | 98 | 84% | B | 102 | 3.65 | 3.63 | SR

56J8 | Gach,T | 75 | 85 | 90 | 83% | B | 60 | 3.75 | 3.72 | SO

78T6 | Johnson,K | 80 | 78 | 89 | 82% | B | 15 | 2.0 | 2.11 | FR

34U8 | Hunter,C | 100 | 50 | 75 | 75% | C | 75 | 2.6 | 2.58 | JR

45P5 | Garrison,J | 75 | 78 | 72 | 75% | C | 39 | 1.85 | 1.85 | SO

44L2 | Levitte,H | 56 | 66 | 99 | 74% | C | 100 | 2.35 | 2.34 | SR

34K5 | Horner,M | 67 | 75 | 23 | 55% | F | 17 | 1.95 | 1.74 | FR

Process finished with exit code 0

1. Class Documentation

**class Student1** implements Comparable

**Student1 Data**

protected int pscore;

protected int[] testScore = new int[3];

protected String id;

protected String name;

protected String letterGrade;

**Student1 Functions**

public Student1()

public Student1(int[] test, String id, String name)

public int getPscore()

public String toString()

public int compareTo(Object o)

**class Student2** implements Comparable

**Student2 Data**

protected int pscore;

protected int[] testScore = new int[3];

protected String id;

protected String name;

protected String letterGrade;

protected int TotalNoHours;

protected float CumulativeGPA;

protected String studentClass;

protected float newStudentGPA;

**Student2 Functions**

public Student2()

public Student2(int[] test, String id, String name, int TotalNoHours, float CumulativeGPA)

public int getPscore()

public String toString()

public int compareTo(Object o)

public **class CodeforProb2CSC285**

**CodeforProb2CSC285 Data**

The following Data is within the main method of this class:

ArrayList<Student2> Academic\_Class2 = new ArrayList<Student2>()

Scanner input = new Scanner(new File("Input2.txt"))

**CodeforProb2CSC285 Functions**

public static void AddStudent(ArrayList<Student2> Academic\_Class2, Student2 Obj)

public static void DeleteStudent(ArrayList<Student2> Academic\_Class2, String StudentID)

public static void SortLarge(ArrayList<Student2> Academic\_Class2)

public static void main(String[] args) throws FileNotFoundException