1. Nick Wright CSC285 Binary Tree Final Project

2. Java Code

/\* This is a program that is capable of reading in student information from an input file, and putting it in a binary tree of student objects.  
 \* The program currently contains the following three classes: MyBinaryTreeMgr, Student2, binaryTreeFinalProject \*/  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.util.Scanner;  
  
/\* class MyBinaryTreeMgr is a generic class that is capable of storing objects  
 \* in a binary tree in order based on their compareTo function.\*/  
class MyBinaryTreeMgr<T extends Comparable>{  
 protected TreeNode<T> root;//this is the root of the tree  
 protected int number;//this is the number of nodes in the tree  
 public MyBinaryTreeMgr(){  
 //this is the constructor for MyLinkManager  
 root=null;//set the root of the tree to null  
 int number = 0;//nothing in the tree  
 }//end of the constructor  
 public int getnumber(){return number;}  
 private static class TreeNode<T extends Comparable>{  
 //this is an internal class for constructing nodes in MyBinaryTreeMgr  
 protected T nodevalue;  
 protected TreeNode<T> left;//left branch of the tree  
 protected TreeNode<T> right;//right branch of the tree  
 public TreeNode(T x){  
 nodevalue=x;  
 left=null;//create a pointer to the next left node but set it to null  
 right=null;//create the pointer to the next right node but set it to null  
 }//end of TreeNode constructor  
 }//end of class TreeNode  
 protected TreeNode<T> CreateNode(T x){  
 //this creates a new tree node  
 return new TreeNode(x);  
 }//end of CreateNode  
  
 //adding a node to the tree. This will be done with MyBinaryTreeMgr  
 public int insertnode (T x){  
 //this adds a node to the binary tree  
 //it must be determined if this is the root node  
 if(root==null){  
 //creating root node  
 root=CreateNode(x);  
 number=1;  
 return number;  
 }else{  
 //there are already nodes in the tree, the current node must be put in the correct place  
 //starting with the parent node  
 TreeNode<T> parent = null;  
 TreeNode<T> current = root;  
 while (current!=null){  
 //follow this tree until we find the last node  
 if(x.compareTo(current.nodevalue)<0){  
 //this value is less than the current, move left in the tree  
 parent=current;  
 current=current.left;  
 }else if(x.compareTo(current.nodevalue)>0){  
 //we are greater than the current value, move right in the tree  
 parent = current;  
 current=current.right;  
 }else{  
 return -99;//this node is a duplicate. they do not get put in the tree but a value of -99 is returned  
 }//end of first if else chain  
 }//end of while loop  
 if(x.compareTo(parent.nodevalue)<0){  
 parent.left=CreateNode(x);  
 }else{  
 parent.right=CreateNode(x);  
 }//node inserted  
 }  
 number++;  
 return number;  
 }//end of insertnode  
  
 public void inorder (PrintWriter outp){  
 //transverses the tree in order and prints the nodes  
 inorder(root,outp);  
 }//end of the helper inorder  
 protected void inorder(TreeNode<T> root, PrintWriter outp){  
 //this is a recursive function to print the nodes of a binary tree in order  
 //this will print the nodes from left-middle-right (scan inorder)  
 //check to see if we are the last node  
 if(root==null){  
 return;  
 }  
 //move left if not here  
 inorder(root.left,outp);  
 //we have moved as far left as we can print  
 //changing print statement to println statement so that it is easier to read the output of the program  
 System.*out*.println(root.nodevalue+" ");  
 outp.println(root.nodevalue+" ");  
 inorder(root.right,outp);  
 }//end of inorder method  
}//end of class MyBinaryTreeMgr  
  
/\* 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 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  
 public float getCumulativeGPA(){return CumulativeGPA;}//a getter method for the cumulative gpa which wil be used in the compareTo function  
  
 @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;  
 }  
  
 //changing compareTo for binary tree assignment, originally compared pscore but will now compare gpa  
 @Override  
 public int compareTo(Object o) { //changing compareTo function for when we use it in method SortLarge  
 if(getCumulativeGPA()>((Student2)o).getCumulativeGPA()){  
 return 1; //returning 1 if object 1 is larger than object 2  
 } else if (getCumulativeGPA()<((Student2)o).getCumulativeGPA()){  
 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 student2 class  
  
/\* public class CodeforProb2CSC285 contains the main method\*/  
public class binaryTreeFinalProject{  
  
 public static void main(String[] args) throws FileNotFoundException { //FileNotFoundException needed for reading the input file  
 PrintWriter outpt;  
 //setting the internal name to an external file through the printwriter  
 outpt = new PrintWriter(new File("csc285binaryTreeOutput.txt"));  
 MyBinaryTreeMgr<Student2> linkstu = new MyBinaryTreeMgr<Student2>(); //creating a binary tree for student2 objects  
  
 Scanner input = new Scanner(new File("Input2.txt"));  
 while(input.hasNext()){ //while loop used to iterate through the input file and assign variables their values  
 //creating variables that will be used to temporarily store the values being read in from input text before they are put into an object  
 int[] test = new int[3];  
 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 workStu2 = new Student2(test, Sid, name, creditHour, gpa); //creating a new Student2 object  
 linkstu.insertnode(workStu2); //storing object in linkstu  
 }//end of while loop  
  
 System.*out*.println("\n" + "----------------------------------------------------------------------------------------------" + "\n" +  
 " Binary Tree"  
 + "\n" + "----------------------------------------------------------------------------------------------");  
 System.*out*.println("List the student objects from the binary tree in left-middle-right order By their GPA.");  
 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");  
  
 //printing out the objects from the binary tree  
 linkstu.inorder(outpt);  
 outpt.close();  
 } //end of main method  
} //end of binaryTreeFinalProject

3. 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

4. Output Files

----------------------------------------------------------------------------------------------

Binary Tree

----------------------------------------------------------------------------------------------

List the student objects from the binary tree in left-middle-right order By their GPA.

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

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

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

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

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

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

Process finished with exit code 0

5. Class Documentation

**class MyBinaryTreeMgr**<T extends Comparable>

**MyBinaryTreeMgr Data**

protected TreeNode<T> root;

protected int number;

**MyBinaryTreeMgr Functions**

public int getnumber()

protected TreeNode<T> CreateNode(T x)

public int insertnode(T x)

public void inorder(PrintWriter outp)

Protected void inorder(TreeNode<T> root, PrintWriter outp)

**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 int getCumulativeGPA()

public String toString()

public int compareTo(Object o)

public **class binaryTreeFinalProject**

class binaryTreeFinalProject contains the main method

**binaryTreeFinalproject Data**

the following data are from the main methods:

PrintWriter outpt;

MyBinaryTreeMgr<Student2> linkstu = new MyBinaryTreeMgr<Student2>();

Scanner input = new Scanner(new File(“Input2.txt”));

**binaryTreeFinalProject Functions**

the following functions are from the main method:

linkstu.inorder(outpt);

outpt.close();