Hi guys,
There are no questions. Look at the output to see what the program must do. Then maybe open the program in Netbeans and start stepping it through from the main method **StudentUI**. Read over all the comments and then Control Click on the methods to see how the program will jump around. Or go to the Main method in these notes **StudentUI** and work through it.

The text file students.txt contains the following data:

Student Program

```
Storm#18#10000#true#P
Bogosi#17#20000#true#P
Frans#15#30000#false#D
Rayman#16#40000#false
Susan#18#25000#false
Daniel#17#10000#true#H
Rossy#16#50000#false#P
Adru#18#10000#true
Jimmy#18#20000#false
```

<u>Example of OUTPUT:</u>

```
Storm  18    true    Adult  R10000.0     (R11000.0)  Prefect
Bogosi 17    true    Minor  R20000.0     (R22000.0)  Prefect
Frans  15    false   Minor  R30000.0     (R33000.0)  Deputy Head
Rayman 16    false   Minor  R40000.0     (R44000.0)
Susan  18    false   Adult  R25000.0     (R27500.0)
Daniel 17    true    Minor  R10000.0     (R11000.0)  Head
Rossy  16    false   Minor  R50000.0     (R55000.0)  Prefect
Adru   18    true    Adult  R10000.0     (R11000.0)
Jimmy  18    false   Adult  R20000.0     (R22000.0)

The total outstanding school fees: 215000.0
```

AFTER SORTING ACCORDING TO FEES:

```
Frans   15      false   Minor  R30000.0      (R33000.0)  Deputy Head
Rayman  16      false   Minor  R40000.0      (R44000.0)
Rossy   16      false   Minor  R50000.0      (R55000.0)  Prefect
Bogosi  17      true    Minor  R20000.0      (R22000.0)  Prefect
Daniel  17      true    Minor  R10000.0      (R11000.0)  Head
Adru    18      true    Adult  R10000.0      (R11000.0)
Jimmy   18      false   Adult  R20000.0      (R22000.0)
Storm   18      true    Adult  R10000.0      (R11000.0)  Prefect
Susan   18      false   Adult  R25000.0      (R27500.0)


Adru's age is 18

The number of boarders: 4
```

AFTER DELETING Frans:

```
Rayman  16      false   Minor  R40000.0      (R44000.0)
Rossy   16      false   Minor  R50000.0      (R55000.0)  Prefect
Bogosi  17      true    Minor  R20000.0      (R22000.0)  Prefect
Daniel  17      true    Minor  R10000.0      (R11000.0)  Head
Adru    18      true    Adult  R10000.0      (R11000.0)
Jimmy   18      false   Adult  R20000.0      (R22000.0)
Storm   18      true    Adult  R10000.0      (R11000.0)  Prefect
Susan   18      false   Adult  R25000.0      (R27500.0)
```

AFTER MAKING Rayman A BOARDER:

```
Rayman 16   true    Minor R40000.0    (R44000.0)
Rossy  16   false   Minor R50000.0    (R55000.0)  Prefect
Bogosi 17   true    Minor R20000.0    (R22000.0)  Prefect
Daniel 17   true    Minor R10000.0    (R11000.0)  Head
Adru   18   true    Adult R10000.0    (R11000.0)
Jimmy  18   false   Adult R20000.0    (R22000.0)
Storm  18   true    Adult R10000.0    (R11000.0)  Prefect
Susan  18   false   Adult R25000.0    (R27500.0)
```

AFTER INCREASING Bogosi's AGE:

```
Rayman 16   true    Minor R40000.0    (R44000.0)
Rossy  16   false   Minor R50000.0    (R55000.0)  Prefect
Bogosi 18   true    Adult R20000.0    (R22000.0)  Prefect
Daniel 17   true    Minor R10000.0    (R11000.0)  Head
Adru   18   true    Adult R10000.0    (R11000.0)
Jimmy  18   false   Adult R20000.0    (R22000.0)
Storm  18   true    Adult R10000.0    (R11000.0)  Prefect
Susan  18   false   Adult R25000.0    (R27500.0)
```

AFTER ADDING A NEW STUDENT:

```
Rayman 16   true    Minor R40000.0    (R44000.0)
Rossy  16   false   Minor R50000.0    (R55000.0)  Prefect
Tony   18   false   Adult R20123.0    (R22135.3)  Prefect
Bogosi 18   true    Adult R20000.0    (R22000.0)  Prefect
Daniel 17   true    Minor R10000.0    (R11000.0)  Head
Adru   18   true    Adult R10000.0    (R11000.0)
```

```
Jimmy 18    false   Adult   R20000.0    (R22000.0)
Storm 18    true    Adult   R10000.0    (R11000.0)  Prefect
Susan 18    false   Adult   R25000.0    (R27500.0)
```

```java
package studentsui;

public class Student {

    // Creating the fields of the object

    private String name;                   // Other names for fields: instance variables,
    private int age;                       // the object's properties or attributes.
    private double fees;                   // <-- The oustanding school fees
    private boolean boarder;               // <-- Is the students a boarder: true or false
    private char prefect;                  // <-- P : Prefect  H : Head boy/girl   D : Deputy head boy/girl

    // The constructor method is used to create (instantiate) the object and assign data to its fields

    public Student(String n, int a, double f, boolean b, char p) {
        name = n;
        age = a;
        fees = f;
        boarder = b;
        prefect = p;
    }

    // The setter methods are used to change the data of an object's specific field

    public void setName(String n) {
        name = n;
    }

    public void setAge(int a) {
        age = a;
    }

    public void setFees(double f) {
        fees = f;
    }

    public void setBoarder(boolean b) {
        boarder = b;
```

```
}

// The getter methods are used to return an object’s data stored in a specific field:

public String getName() {
    return name;
}

public int getAge() {
    return age;
}

public double getFees() {
    return fees;
}

public boolean getBoarder() {
    return boarder;
}

// The toString method is used to add the data of an object’s fields together into a String and return it.

public String toString() {
    return name + "\t" + age + "\t" +boarder + "\t" + determineStatus() + "\tR" + fees
                                                    + "\t(R"+ ifFeesIncreased()+ ")  " + getLeader();
}

// Instead of adding P, D or H to the toString method we want to add words.
// So in the toString method we make a call to this method that will return
// the words: Prefect, Deputy Head or Head.

public String getLeader() {
    String lead = "";

    if (prefect == 'P') {
        return "Prefect";
    }
                                    // <-- place else's between all the if statements
    if (prefect == 'D') {           //     and click on Format and then Source
        return "Deputy Head";       //     to correct the indentation.
    }

    if (prefect == 'H') {
            return "Head";
        }

        return lead;
    }



    // Returns the school fees increased by 10% to be displayed in the toString method.

    public double ifFeesIncreased() {
        return fees + fees * 10 / 100;
    }

    // Returns the string “Minor” when the object’s age is less than 18 or “Adult” when its age is 18 and above
    // To be used in the toString method.

    public String determineStatus() {
        if (age < 18) {
            return "Minor";
        } else {
            return "Adult";
        }
    }

}
```

```
package studentsui;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

public class StudentManager {

    private Student[] pupil = new Student[100];
```

```
// It creates in memory (without the data in them, the data will be assigned to the field's in the contructor
   method):
//
//     *pupil[0].name: "Storm"
//     *pupil[0].age: 18
//     *pupil[0].fees: 10000.0
//     *pupil[0].boarder: false
//     *pupil[0].prefect: P
//
//     *pupil[1].name: "Bogosi"
//     *pupil[1].age: 17
//     *pupil[1].fees: 20000.0
//     *pupil[1].boarder: true
//     *pupil[1].prefect: P
//
//     *pupil[2].name:
//     *pupil[2].age:
//     *pupil[2].fees:
//     *pupil[2].boarder:


 private int size = 0;                              //   *size: 0

// In the constructor method we read the data out of the text file line by line and
// create objects (for each line) and assign values to the object's fields.

public StudentManager() {

    try {
        Scanner scFile = new Scanner(new File("Students.txt"));

        // Let's say the Students.txt text file contains the following data:
        // Storm#18#10000#true#P
        // Bogosi#17#20000#true#P
        // Frans#15#30000#false#D
        // Rayman#16#40000#false
        // Susan#18#25000#false
        // Daniel#17#10000#true#H
        // Rossy#16#50000#false#P
        // Adru#18#10000#true
        // Jimmy#18#20000#false

        while (scFile.hasNext()) {        // while there is still something in the file repeat...
```

```
String line = scFile.nextLine();       // Read a line of text from the text file and store it in the
                                          line variable.

                                       // E.g. *line: Storm#18#10000#true#P





// The next section will split the line of text up in to different parts and store them in variables.

Scanner scLine = new Scanner(line).useDelimiter("#");  // We say that the # character seperates the
                                                          data.

String names = scLine.next();          // Reads out the string out of the line variable and assign it
                                          to names.
                                       // E.g. *names: Storm

int years = scLine.nextInt();          // Reads out the integer out of the line variable and assign it
                                          to years.
                                       // E.g. *years: 18

double due = scLine.nextDouble();      // Reads out the double out of the line variable and assign it
                                          to due.
                                       // E.g. *due: 10000.00

boolean hostel = scLine.nextBoolean(); // E.g. *hostel: true

char leader = ' ';                     // NB: Create varaible before the if statement!!! NOT inside;

if (scLine.hasNext()) {                // NB: Not everyone has a P/D/H in the text file - so get the
                                           next thing from the line variable only when there is
                                           something next in it.
    leader = scLine.next().charAt(0);  //  Reads out the character out of the line variable and assign
                                           it to leader
}

// The reads must be in the same order as the order of the data in the line variable.
// Next we are going to send above data to the contructor method to create an object (e.g. pupil[0] )
// and assign the values to the object's fields:
```

```
            pupil[size] = new Student(names, years, due, hostel, leader);

                                              // It sends "Storm", 18, 10000.0, etc. to constuctor method

            // It reads as follows: pupil[ 0 ] = new Student("Storm", 18, 10000, true); since size is 0.
            // The constructor method creates the object pupil[0] and assign above data to it's fields:
            //     *pupil[0].name: "Storm"
            //     *pupil[0].age: 18
            //     *pupil[0].fees: 10000.0
            //     *pupil[0].boarder: true
            //     *pupil[0].prefect: 'P'
            //

            //      The first object is called pupil[0].
            //      The second object will be called pupil[1], etc.

            size++;                           // Increase size by 1, so next time it creates a new object,
                                              // the object will be pupil[1].

        }       //Jumps up to the above where statement.

    } catch (FileNotFoundException ex) {
        JOptionPane.showMessageDialog(null, "Error! File not found"); //Displays this message when the file was not
                                                                        found.
    }

}

// This method will always be asked - you need to display all the data in the array of objects. Here you add
// all the data to a String variable called output, using the toString method to return all the data of each
// object.

public String listAllStudents() {       //We are returning one big String.

    String output = "";                 // Create the output variable.
                                        // *output: ""

    for (int i = 0; i < size; i++) {                 // Loop through the whole array. i will first become 0.  *i: 0
        output = output + pupil[i].toString() + "\n";   //    pupil[0].toString() means call (go to) the toString()
                                                              method.
                                                        //    The toString method combine all the data of the first
                                                              object pupil[0] together and returns it as one
                                                              String. It then gets added to output.
```

```
                                          //    *output: "Storm\t18\ttrue\tAdult\tR10000.0\t(R11000.0)\tPrefect
        }                                //    It jumps up to the for loop. i increases by 1.  *i: 1
                                        //    pupil[1].toString() will return the second object pupil[1]'s data as
                                                    //    one string and add it to output.
                                       //    *output: "Storm\t18\ttrue\tAdult\tR10000.0\t(R11000.0)\tPrefect\n
                                                    Bogosi\t17\ttrue\tMinor\tR20000.0\t(R22000.0)\tPrefect
                                     //    etc.....

        return output;                                  // returns the data stored in the  output variable.

            //NB: To use methods from the first object class we need to use the
            //    format: object.method()
            //       E.g: pupil[2].toString();
            //           doulbe amount = pupil[9].getFees();  //getter methods needs to be assigned to a variable.
            //           pupil[0].setFees(5);
    }

    // In this method we calculate the total amount of outstanding school fees and return it.

    public double totalOutstandingFees() {
        double amount = 0;

        for (int i = 0; i < size; i++) {
            amount = amount + pupil[i].getFees();       // Adds pupil[0].getFees (which is 10000.0) to amount.
                                                        // Next, adds pupil[1].getFees (which is 20000.0) to amount.
        }                                               // etc....

        return amount;                                  // E.g. returns 20000.0
    }

// The next method will sort the array first according to the age, then there where the ages are the
// same we sort the names in alphabetic order. Please note that we cannot say pupil[i].getName() > pupil[j].getName()
// to compare them but use the compareTo method (SEE CHARACTER & STRING NOTES where it is explained)

    public void sortAgeName() {   //the return type is void because the method contains no return statement

        for (int i = 0; i < size - 1; i++) {

            for (int j = i + 1; j < size; j++) {

                if (pupil[i].getAge() > pupil[j].getAge()) {   // Ascending > and Descending <
                    Student temp;                           // Because we are swapping objects the data type of temp is
                    temp = pupil[i];                // our object class Student. It will create *temp.name:  *temp.age:
                    pupil[i] = pupil[j];           // and  *temp.fees: etc...
                    pupil[j] = temp;
                } else if (pupil[i].getAge() == pupil[j].getAge()) {        // there where the ages are the same we
                                                                              check the names...
                    if (pupil[i].getName().compareTo(pupil[j].getName()) > 0) {
                        Student temp;
                        temp = pupil[i];
                        pupil[i] = pupil[j];
                        pupil[j] = temp;
                    }
                }

            }

        }
    }

    // The next method is a search - it will get a name as a parameter, search for that name in the array and return
       the person's age

    public int getStudentsAge(String findName) {   // It creates the findName variable and places the name in it that
                                                      was sent to it

        int old = -1;                     // Create a variable where I can place the result in. I make it -1 because if
                                            the name is not found it will return -1 (which will indicate that it was
                                            not found).

        boolean found = false;           // We asume that we are not going to find it.   *found: false

        for (int i = 0; i < size; i++) {    //Loop throught the array and try and find the name

            if (pupil[i].getName().equals(findName)) {

                                          // If the name has been found. Cannot use: pupil[i].getName() == findName

                found = true;                               // sets found to true
                old = pupil[i].getAge();                    // Get that student's age
                break;                                      // It is used to exit the loop as soon as it is found.
            }

        }
        return old;                    // Return the result - the age.

    }

    // The next method will count how many boarders there are and return the number

    public int countBoarders() {

        int count = 0;

        for (int i = 0; i < size; i++) {      // This time we cannot use a while loop since we need to check every age
                                                 in the array
            if (pupil[i].getBoarder() == true) {   //If in the range ... OR JUST: if ( pupil[i].getBoarder() )
                count++;                            // ... then increase count by 1
            }
        }

        return count;
    }

    // The following method we are going to use to DELETE a student from the array.
    // EXAMPLE: Let's say the array is as follow:
    //    pupil[0] -> Storm    pupil[1] -> Bogosi    pupil[2] -> Frans    pupil[3] --> Rayman    pupil[4] --> Susan
    // We want to remove Frans from the array.
    // So what we will do is shift everything UP at a certain position.
    // In this case it will be the position (index) 3, since Rayman (at position 3) will need to move into Frans's
       place.
    // After the shifting has taken place, the array will look as follow:
    //    pupil[0] -> Storm    pupil[1] -> Bogosi    pupil[2] -> Rayman    pupil[3] --> Susan ....

    public void deleteStudent(String searchName) {          // Receive the name of the student I want to delete.
                                                                  *searchName: Frans

        //(A) First we go and search for the position of the studName.

        int position = -1;

        for (int i = 0; i < size; i++) {

            if (pupil[i].getName().equals(searchName)) {

                                                  //OR in two instructions: (1) String studName = pupil[i].getName();
                position = i;                 //                         (2) if (studName.equals(searchName)
                break;
            }

        }

        // (B) Next we SHIFT everything one place UP from that position

        if (position != -1)               // In other words: If the name was found (the position is not -1).
        {

            for (int i = position; i < size - 1; i++) {   // It will shift all the objects one place UP left from
                                                             the certain position
                pupil[i] = pupil[i + 1];                 // SEE ABOVE EXAMPLE
            }

            size--;                      //NB: There is one less student, so the array has decreased by 1!!!
        }

    }


    // In the next method we are going to make a student a boarder - change his/her boarder field to true

    public void becomeBoarder(String searchName)     // Lets say Rayman goes to boarding house. His name was sent here.
    {                                                          // *searchName: Rayman

        //(A) First we go and search for the position of Rayman in the array.

        int position = -1;

        for (int i = 0; i < size; i++) {          // Loop through array - Is Storm = Rayman? No  (i was 0)
                                                  //                       Is Bogosi = Rayman? No    (i was 1)
                                                  //                       Is Rayman = Rayman? Yes   (i was 2)
                                                  //        Then make position 2 - Rayman's array index is 2)

            if (pupil[i].getName().equals(searchName)) {       //OR in two instructions:
                                                                        (1) String studName = pupil[i].getName();
                position = i;                                 //       (2) if (studName.equals(searchName)
                break;
            }

        }
        // (B) Next we use a setter method to change the student's boarder field to true

        if (position != -1) {                   // In other words: If the name was found (the position is not -1).
            pupil[position].setBoarder(true);   // Thus: pupil[2].setBoarder(true); will send the value true to the
                                                         setBoarder method
        }
                                      // The setBoarder method will receive the true (and place it in b) and change the
                                      // object's field's value by using the instruction: boarder = b;
                                      // Thus chaging the fields boarder value to true:  *pupil[2].boarder: true
    }
```

```
// In the next method we are going to change a student's age.

public void changeAge(String searchName, int newAge)

                                            // Lets say Bogosi turns 18. The method receives two parameters:
{                                           // The name to be searched (e.g. Bogosi) and the new age (e.g. 18)
    // Thus: *searchName: Bogodir  *newAge: 18

    //(A) First we go and search for the position of Bogosi in the array.

    int position = -1;

    for (int i = 0; i < size; i++) {               // Loop through array - Is Storm = Bogosi? No  (i was 0)
                                                   //                        Is Bogosi = Bogosi? Yes  (i was 1)
                                                  //  Then make position 1  (Bogosi's array index is 1)

        if (pupil[i].getName().equals(searchName)) {

                                    //OR in two instructions: (1) String studName = pupil[i].getName();
            position = i;           //                        (2) if (studName.equals(searchName)
            break;
        }

    }

    // (B) Next we use a set method to change the student's boarder field to true

    if (position != -1)              // In other words: If the name was found (the position is not -1).
    {
        pupil[position].setAge(newAge);   // Thus: pupil[1].setAge(18); will send the value 18 to the setAge method
    }
                                          // The setAge method will receive the 18 (and place it in a) and change the
                                         // object's field's value by using the instruction: age = a;;
                                        // Thus chaging the fields age value to 19:  *pupil[1].age: 18
}

// The following method we are going to use to INSERT a student into the array at a certain position.
// EXAMPLE: Let's say the array is as follow:
//    pupil[0] -> Storm    pupil[1] -> Bogosi    pupil[2] -> Rayman    pupil[3] --> Susan  (Frans was deleted)
// We want to insert the new student (say Tony) before Bogosi.
// So what we will do is shift everything (from Keene onwards) one place DOWN to make a place for Tony.
// The shifting will start at a certain position. You need to find this position first.
// After the shifting has taken place, the array will look as follow:
//    pupil[0] -> Storm    pupil[1] -> _____ pupil[2] -> Bogosi    pupil[3] -> Rayman    pupil[4] --> Susan
// Now we can insert Tony there.
public void insertStudent(String searchName, Student newStud) {

                                    // Receive the name of a student (that is in the array) BEFORE which I want to
                                  // insert the new student, e.g. Bogosi
                                // Also receive the new students information (as an object).
                              // *searchName: Bogosi
                            // *newStud.names: Tony  *newStud.years: 18  *newStud.due: 20123  *newStud.hostel: false

    //(A) First we go and search for the position of the studName.

    int position = -1;

    for (int i = 0; i < size; i++) {

        if (pupil[i].getName().equals(searchName)) {          //OR in two instructions:
                                                                    (1) String studName = pupil[i].getName();
            position = i;                                 //        (2) if (studName.equals(searchName)
            break;
            }

        }

        // (B) Next we SHIFT everything one place DOWN at that position

        if (position != -1)                     // In other words: If the name was found (the position is not -1).
        {

            for (int i = size; i >= position; i--) {   // It will shift all the objects one place DOWN from the certain
                                                          position
                pupil[i] = pupil[i - 1];               // opening a place for the new student. SEE ABOVE EXAMPLE
            }

            pupil[position] = newStud;      // Thus it reads as: pupil[1] = newStudent;  where newStud is the parameter
                                           // which is an object where the data was stored that was sent to this method.
        }

        size++;           //We have one new student, so the array has increased by 1 !!!

    }

}
```

```
package studentsui;

import javax.swing.JOptionPane;


public class StudentsUI {


    public static void main(String[] args) {

        // To be able to use the methods of the StudentManager class we need to create
        // an object of that class and then say: object.method();
```

```
StudentManager sm = new StudentManager();

// We can split above instruction into 2 parts:
// (1) StudentManager sm;  Whenever it sees a class it will go to that class and create the variable (fields)
//                         listed there at the top of the class. So it will create the array at the top of the
//                         StudentManager class. It will also create the size variable to store the size of the
                           array.
// (2) sm = new StudentManager();  This will run the constructor method StudentManager() - it will go there
//                                 and execute all the code in that method.
//                                 It will read the lines of text from the file, create objects and place
                                   (assign)
//                                 the data to the fields/variables of all the objects.

// To display all the objects' data (all the students' info) we make a call to the listAllStudents() method:

System.out.println(sm.listAllStudents());   // again: object.method  - sm is the object and listAllStudents is
                                               the method
                                            // It needs to be in a println since it returns data - a String

// Next, display the total outstanding school fees:

double outstanding = sm.totalOutstandingFees();

System.out.println("The total outstanding school fees: " + outstanding);

         // or just: System.out.println("The total outstanding school fees: " + sa.totalOutstandingFees() );



// Next, we sort the students acording to their ages and where the ages are the same - we sort the names
    (see output)

sm.sortAgeName();

// We display the data again with a heading:

System.out.println("\nAFTER SORTING ACCORDING TO FEES:\nNAME\tAGE\tFEES\tBOARDER\n" + sm.listAllStudents());

// Next the user wants to know someones age. He will enter the name and the person's age will be displayed.
// Here we just send the name Adru to the method.

int age = sm.getStudentsAge("Adru");
if (age == -1) {
    System.out.println("\nAdru was not found");
} else {
    System.out.println("\nAdru's age is " + age);
}

// Next we want to display how many students are boarders

System.out.println("\nThe number of boarders: " + sm.countBoarders());    // Done this time in one instruction.


// Next we want to delete a student:
// First enter the student's name that I want to delete. Then make a call to the deleteStudent method.
// Here we just delete Frans (because he left the school)

sm.deleteStudent("Frans");

// Lets display the students' info:

System.out.println("\n\nAFTER DELETING Frans:\n" + sm.listAllStudents());

// Next we want to make a student a boarder. So change a student's boarder field to true.
// Rayman is going to boarding.

sm.becomeBoarder("Rayman");

//Lets display the students' info:

System.out.println("\n\nAFTER MAKING Rayman A BOARDER:\n" + sm.listAllStudents());


// Next we want to change a students age. We will enter the student's name and age and send that as parameters
// to the changeAge method. Here we just change Bogosi's age from 17 to 18 because it was his Birthday

sm.changeAge("Bogosi",18);

//Lets display the students' info:

System.out.println("\n\nAFTER INCREASING Bogosi's AGE:\n" + sm.listAllStudents());


// Next we want to insert a new student. We will enter his/her information here and send it to the
   insertStudent method
// .....
```

```
        Student stud = new Student("Tony",18,20123,false,'P');  // We want to send the data as an object to the method,
                                                                   so we
                                                                // create an object for the new student.
                                                                // This will create the following in memory:
                                                                //     *stud.name: "Tony"
                                                                //     *stud.age: 18
                                                                //     *stud.fees: 20123.0
                                                                //     *stud.boarder: false
                                                                //     *stud.prefect: 'P'

        sm.insertStudent("Bogosi", stud);     // We want to enter the Tony's object (called stud) BEFORE Bogosi's
                                                 object.
                                              // We send the whole object stud (look at the variables above) to the
                                                 insertStudent method

        //Lets display the students' info:

        System.out.println("\n\nAFTER ADDING A NEW STUDENT:\n" + sm.listAllStudents());


    }

}
```