Luis Gutierrez

CS 152

Meeting Time: MW 1:40 - 3:00pm

Student.java

```
/**
 * Write a description of class Student here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Student implements Comparable<Student>
{
    // instance variables - replace the example below with your own
    private int age;
    private double gpa;
    private String lastName;

    /**
     * Constructor for objects of class Student
     */
    public Student(String s, double g, int a)
    {
        // initialise instance variables
        lastName = s;
        gpa = g;
        age = a;
    }

    /**
     * Get the GPA of this Student
     * @return  The GPA of the student
     */
    public double getGPA()
    {
        return gpa;
    }

    /**
     * Determine whether this student is an honor student
     * @return  true or false
     */
    public boolean isHonors()
    {
        if (gpa >= 3.5)
        {
            return true;
```

```java
        }
        return false;
    }

    /**
     * Create a string representation of this student
     * @return  A string
     */
    public String toString()
    {
        String rep = "Last Name: %s\tAge: %d\tGPA: %.2f";
        return String.format(rep, lastName, age, gpa);
    }

    /**
     * Compares this student's GPA against another student
     * @return The integers -1, 0, or 1 if this student's GPA is less than, equal to,
or greater than the other students, respectively
     */
    @Override
    public int compareTo(Student s){
        if (gpa < s.getGPA())
        {
            return -1;
        }
        else if (gpa > s.getGPA())
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}
```

LinkedListStud.java

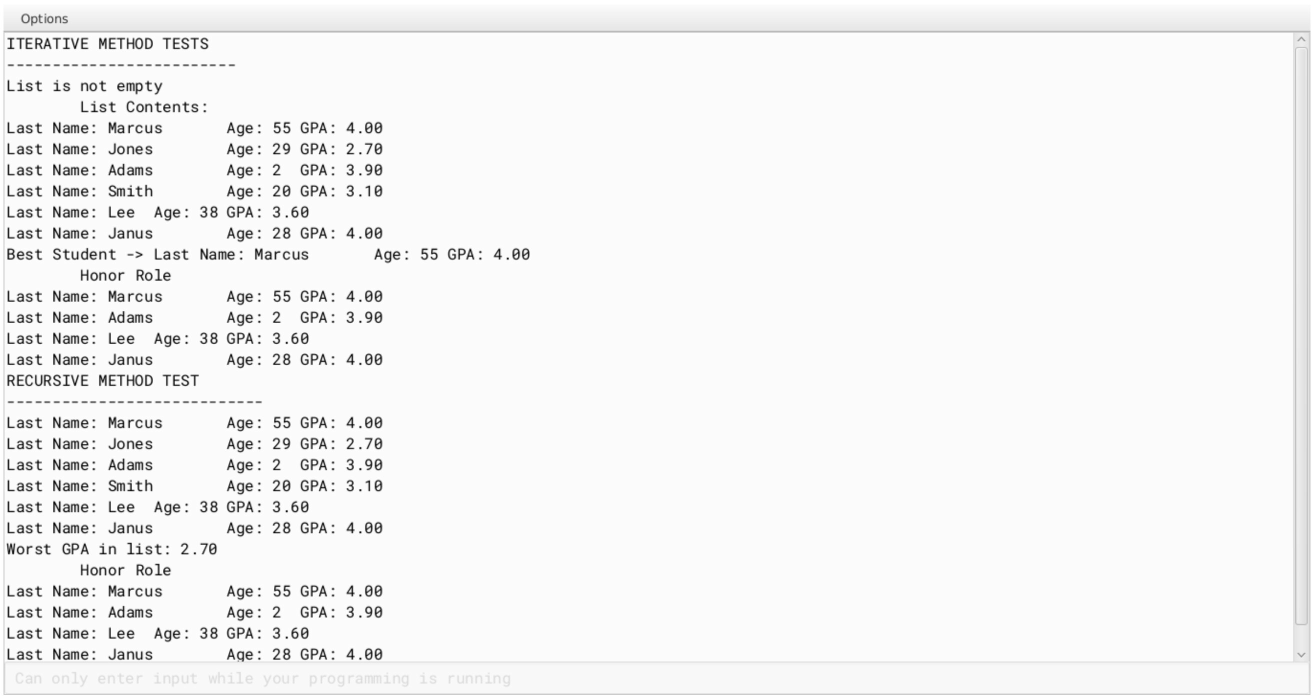
```java
import java.util.ArrayList;
/**
 * Write a description of class LinkedListStud here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class LinkedListStud
{
    // instance variables - replace the example below with your own
    private Node list;

    /**
     * Constructor for objects of class LinkedListStud
     */
    public LinkedListStud()
    {
        // initialise instance variables
        list = null;
    }

    /**
     * Returns reference to head of the list
     * @return  the head of the list
     */
    public Node getList()
    {
        return list;
    }

    /**
     * Checks to see if the list is empty
     * @return true or false
     */
    public boolean isEmpty()
    {
        if (list == null)
        {
            return true;
        }
        return false;
    }

    /**
     * Appends students to head of the list
     * @param   s Student object to be appended to the front
     */
    public void addFront(Student s)
    {
        Node head = new Node(s);

        if (isEmpty())
        {
            list = head;
        }
        else
        {
            head.next = list;
            list = head;
        }
    }

    /**
     * Appends student to end of the list
     * @param   s The student object to be appended to the end of the list
     */
    public void addTail(Student s)
```

```java
    {
        Node tail = new Node(s);
        if(isEmpty())
        {
            list = tail;
        }
        else
        {
            Node curr = list;
            boolean found = false;
            while(!found)
            {
                if (curr.next == null)
                {
                    curr.next = tail;
                    found = true;
                }
                else
                {
                    curr = curr.next;
                }
            }
        }
    }

    /**
     * Prints string representation for each student in the list on a new line to
standard output
     */
    public void printLinkedList()
    {
        if (!isEmpty())
        {
            Node curr = list;
            do
            {
                System.out.println(curr.data);
                curr = curr.next;
            } while (curr != null);
        }
    }

    /**
     * Finds the student with the best GPA
     * @return  Student object with the best GPA
     */
    public Student bestStudent()
    {
        if(!isEmpty())
        {
            Node best = list;
            Node next = list.next;
            while(next != null)
            {
                if (best.data.getGPA() < next.data.getGPA())
                {
                    best = next;
                }
                next = next.next;
            }
            return best.data;
        }
        else
            return null;
    }

    /**
     * Creates an ArrayList containing all honor students
     * @return An ArrayList of Student objects
     */
    public ArrayList<Student> honorsStudents()
    {
        if(!isEmpty())
        {
            ArrayList<Student> ls = new ArrayList();
            Node curr = list;
            while(curr != null)
            {
                if (curr.data.isHonors())
                {
                    ls.add(curr.data);
                }
                curr = curr.next;
            }
            return ls;
        }
        else
            return new ArrayList();
    }

    /**
     * Recursively prints string representation for each student in the list on a new
line to standard output
     * @param   first The starting point in a linked list from which to iterate from
     */
    public void printListRec(Node first)
    {
        if (first != null)
        {
            System.out.println(first.data);
            printListRec(first.next);
        }
    }

    /**
     * Recursively search for the worst GPA in the list
     * @param   first The starting point in a linked list from which to iterate from
     * @return  The lowest GPA in the list
     */
    public double worstGpaRec(Node first)
    {
        if (first != null)
        {
            double gpa = first.data.getGPA();
            if (gpa <= worstGpaRec(first.next))
            {
                return gpa;
            }
            else
            {
                return worstGpaRec(first.next);
            }
        }
        else
            // if null the list has reached the end
            // and smallest value is in a previous stack
            // that value should be less than 100
            return 100;
    }

    /**
     * Recusively creates an ArrayList of all honors students
     * @param   first The starting point in a linked list from which to iterate from
     * @return  An ArrayList of Student objects
     */
    public ArrayList<Student> honorsStudentsRec(Node first)
    {
        if (first != null)
        {
            ArrayList<Student> ls = new ArrayList();
            if (first.data.isHonors())
            {
                ls.add(first.data);
            }
            ls.addAll(honorsStudentsRec(first.next));
            return ls;
        }
        else
        {
            return new ArrayList();
        }
    }

    /**
     * Struct-like inner class representing list items
     */
    private class Node
    {
        public Student data;
        public Node next;
        /**
         * Constructor for objects of class Node
         */
        public Node(Student s)
        {
            data = s;
            next = null;
        }
    }

}
```

TestList.java

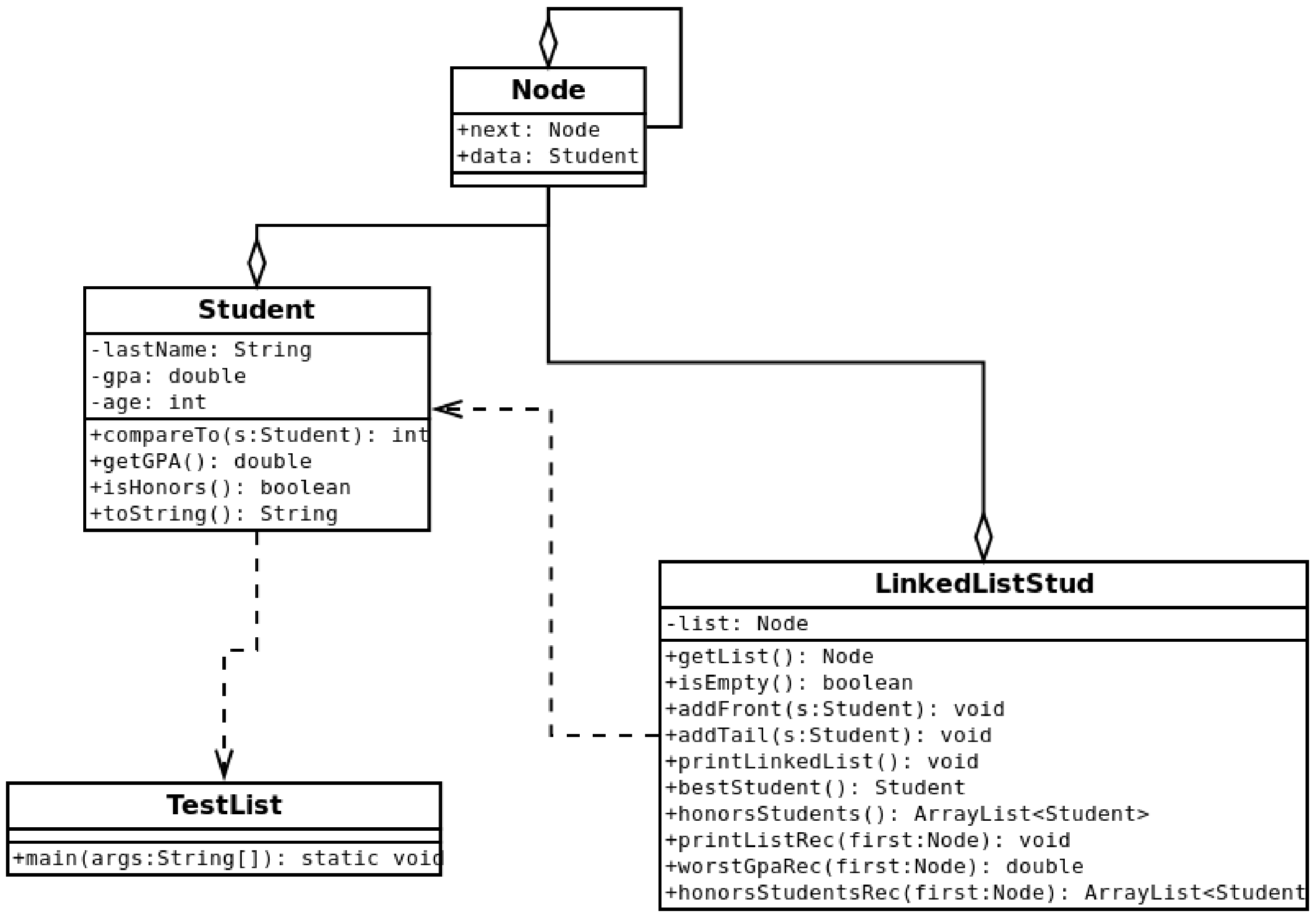
```

/**
 * Write a description of class TestList here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class TestList
{
    public static void main(String[] args)
    {
        LinkedListStud ll = new LinkedListStud();

        ll.addFront(new Student("Adams", 3.9, 2));
        ll.addFront(new Student("Jones",2.7,29));
        ll.addFront(new Student("Marcus",4,55));

        ll.addTail(new Student("Smith",3.1,20));
        ll.addTail(new Student("Lee",3.6,38));
        ll.addTail(new Student("Janus", 4, 28));

        System.out.println("ITERATIVE METHOD TESTS\n-------------------------");
        if (ll.isEmpty())
        {
            System.out.println("List is empty");
        }
        else
        {
            System.out.println("List is not empty");
        }
        System.out.println("\tList Contents:");
        ll.printLinkedList();
        String bestStudentMsg = String.format("Best Student -> %s", ll.bestStudent());
        System.out.println(bestStudentMsg);
        System.out.println("\tHonor Role");
        for(Student s : ll.honorsStudents())
        {
            System.out.println(s);
        }
        System.out.println("RECURSIVE METHOD TEST\n----------------------------");
        ll.printListRec(ll.getList());
        String worstGpaMsg = String.format("Worst GPA in list: %.2f",
ll.worstGpaRec(ll.getList()));
        System.out.println(worstGpaMsg);
        System.out.println("\tHonor Role");
        for(Student s : ll.honorsStudentsRec(ll.getList()))
        {
            System.out.println(s);
        }
    }
}
```

Output



```
ITERATIVE METHOD TESTS
-------------------------
List is not empty
        List Contents:
Last Name: Marcus       Age: 55 GPA: 4.00
Last Name: Jones        Age: 29 GPA: 2.70
Last Name: Adams        Age: 2  GPA: 3.90
Last Name: Smith        Age: 20 GPA: 3.10
Last Name: Lee  Age: 38 GPA: 3.60
Last Name: Janus        Age: 28 GPA: 4.00
Best Student -> Last Name: Marcus       Age: 55 GPA: 4.00
        Honor Role
Last Name: Marcus       Age: 55 GPA: 4.00
Last Name: Adams        Age: 2  GPA: 3.90
Last Name: Lee  Age: 38 GPA: 3.60
Last Name: Janus        Age: 28 GPA: 4.00
RECURSIVE METHOD TEST
----------------------------
Last Name: Marcus       Age: 55 GPA: 4.00
Last Name: Jones        Age: 29 GPA: 2.70
Last Name: Adams        Age: 2  GPA: 3.90
Last Name: Smith        Age: 20 GPA: 3.10
Last Name: Lee  Age: 38 GPA: 3.60
Last Name: Janus        Age: 28 GPA: 4.00
Worst GPA in list: 2.70
        Honor Role
Last Name: Marcus       Age: 55 GPA: 4.00
Last Name: Adams        Age: 2  GPA: 3.90
Last Name: Lee  Age: 38 GPA: 3.60
Last Name: Janus        Age: 28 GPA: 4.00
```

UML


Node
+next: Node
+data: Student
Student
-lastName: String
-gpa: double
-age: int
+compareTo(s:Student): int
+getGPA(): double
+isHonors(): boolean
+toString(): String
LinkedListStud
-list: Node
+getList(): Node
+isEmpty(): boolean
+addFront(s:Student): void
+addTail(s:Student): void
+printLinkedList(): void
+bestStudent(): Student
+honorsStudents(): ArrayList<Student>
+printListRec(first:Node): void
+worstGpaRec(first:Node): double
+honorsStudentsRec(first:Node): ArrayList<Student
TestList
+main(args:String[]): static void