Bachelor in Informatics Engineering	Algorithms and Data Structures 29th of June, 2015. EXTRAORDINARY CALL

Name: ....

Surname: ....

PLEASE, READ CAREFULLY THE FOLLOWING INSTRUCTIONS BEFORE START

1. All student data asked in this document must be filled. Please, use a pen to write in this document.
2. This document and all additional sheets must be delivered upon completion.
3. You cannot leave the classroom until the exam has ended.
4. Mobile phones must be disconnected during the exam.
5. **In this exam, the use of EDALib classes is NOT allowed. You cannot use interfaces and classes provided by the Java API (ArrayList, LinkedList, etc.). The use of Arrays is also forbidden. That is to say, in your solutions you must use your own data structures.**

DO NOT GO BEYOND THIS PAGE until the exam has started.

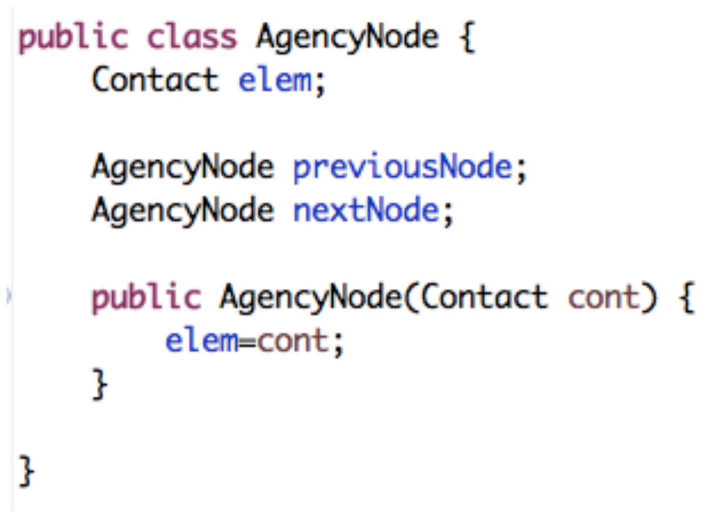
Problem 1 (4 points). Given the following class:

```java
public class Contact {
      String name;
      String email;
}
```

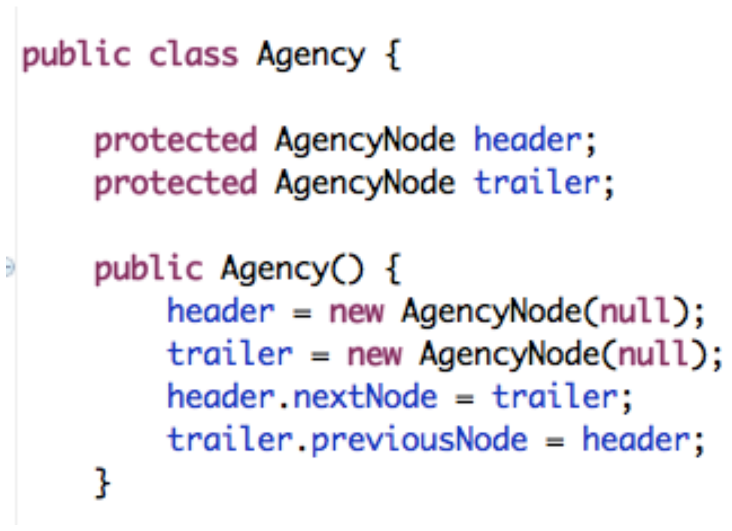
Note: It is recommended to read every paragraph in this exercise before start.

Answer the following:

a) (0.5 points) Implement a **linear structure**, Agenda, storing all contacts in a contact list. In this section, you ONLY have to write headers for classes, attributes and constructors (if you consider them adequate) needed to implement a linear structure. **Explain the election of the type of linear structure you have chosen to implement your solution.**



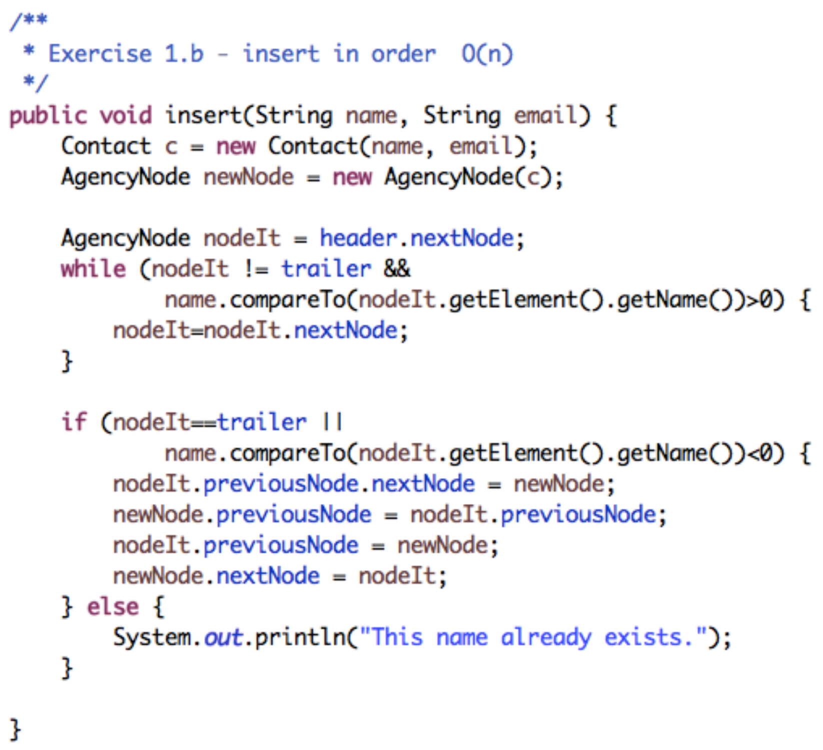
```java
public class AgencyNode {
    Contact elem;

    AgencyNode previousNode;
    AgencyNode nextNode;

    public AgencyNode(Contact cont) {
        elem=cont;
    }

}
```



```java
public class Agency {

    protected AgencyNode header;
    protected AgencyNode trailer;

    public Agency() {
        header = new AgencyNode(null);
        trailer = new AgencyNode(null);
        header.nextNode = trailer;
        trailer.previousNode = header;
    }
```

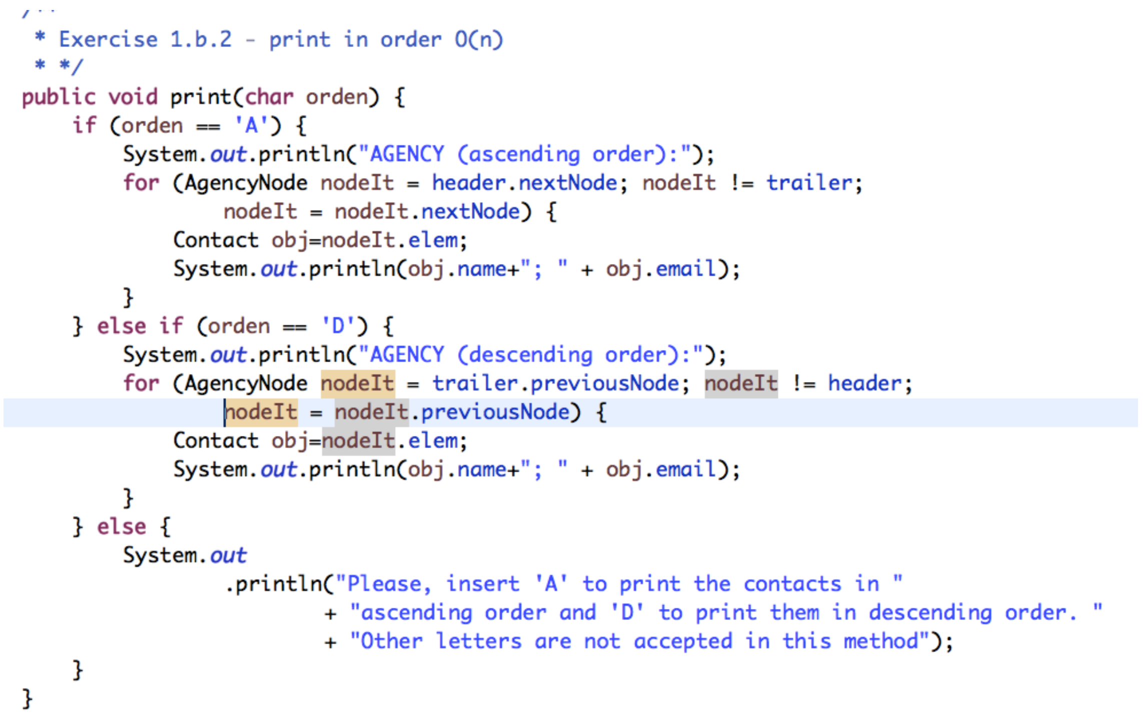
b) Code the following methods:

   i. (1.5 points) Insert in alphabetical order (ascending). This method gets a name and an email at the input and stores them as a contact in the contact list according to an alphabetical order for the name attribute. Duplicates are not allowed.



```
/**
 * Exercise 1.b - insert in order  O(n)
 */
public void insert(String name, String email) {
    Contact c = new Contact(name, email);
    AgencyNode newNode = new AgencyNode(c);

    AgencyNode nodeIt = header.nextNode;
    while (nodeIt != trailer &&
            name.compareTo(nodeIt.getElement().getName())>0) {
        nodeIt=nodeIt.nextNode;
    }

    if (nodeIt==trailer ||
            name.compareTo(nodeIt.getElement().getName())<0) {
        nodeIt.previousNode.nextNode = newNode;
        newNode.previousNode = nodeIt.previousNode;
        nodeIt.previousNode = newNode;
        newNode.nextNode = nodeIt;
    } else {
        System.out.println("This name already exists.");
    }

}
```

   ii. (1.5 points) Show contact list contents in alphabetical order. This method takes a char parameter and, according to this parameter, shows contacts in ascending ('A') or descending ('D') order.



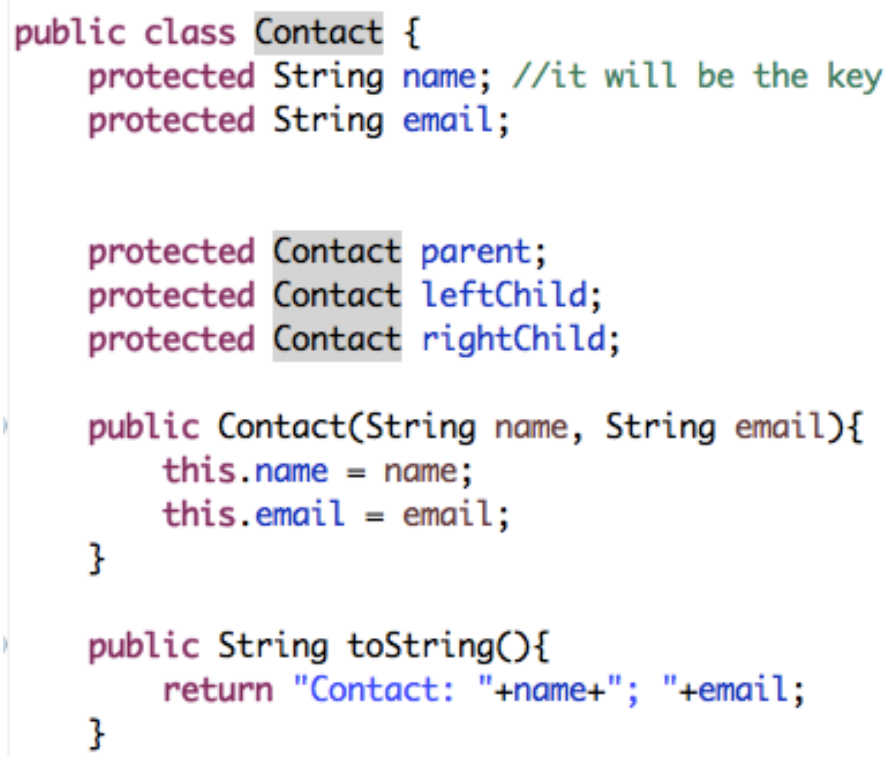
```
/**
 * Exercise 1.b.2 - print in order O(n)
 * */
public void print(char orden) {
    if (orden == 'A') {
        System.out.println("AGENCY (ascending order):");
        for (AgencyNode nodeIt = header.nextNode; nodeIt != trailer;
                nodeIt = nodeIt.nextNode) {
            Contact obj=nodeIt.elem;
            System.out.println(obj.name+"; " + obj.email);
        }
    } else if (orden == 'D') {
        System.out.println("AGENCY (descending order):");
        for (AgencyNode nodeIt = trailer.previousNode; nodeIt != header;
                nodeIt = nodeIt.previousNode) {
            Contact obj=nodeIt.elem;
            System.out.println(obj.name+"; " + obj.email);
        }
    } else {
        System.out
                .println("Please, insert 'A' to print the contacts in "
                        + "ascending order and 'D' to print them in descending order. "
                        + "Other letters are not accepted in this method");
    }
}
```

c) (0.5 points) What is the complexity of each of the methods in b.) section?

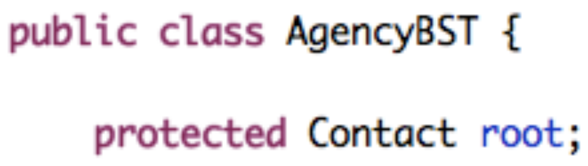
Note: Remember that you cannot use EdaLib and, consequently, you must implement any auxiliary method needed in your code.

Problem 2 (5 points). Considering Contact class provided in the former exercise, solve the following questions:

a) (0.50 points) Implement a class to store all contacts in a Binary Search Tree (BST). This structure will store contacts in an alphabetical order (according to the name). Duplicates are not allowed. If needed, modify the class Contact to include new attributes related with the structure that you could need to implement your solution. In this section, you ONLY have to write headers for classes, attributes and constructors (if you consider them adequate). So, you must not implement any method for now. Which are the advantages of using a BST tree instead of a linear structure to store contacts in alphabetical order?



```java
public class Contact {
    protected String name; //it will be the key
    protected String email;


    protected Contact parent;
    protected Contact leftChild;
    protected Contact rightChild;

    public Contact(String name, String email){
        this.name = name;
        this.email = email;
    }

    public String toString(){
        return "Contact: "+name+"; "+email;
    }
```

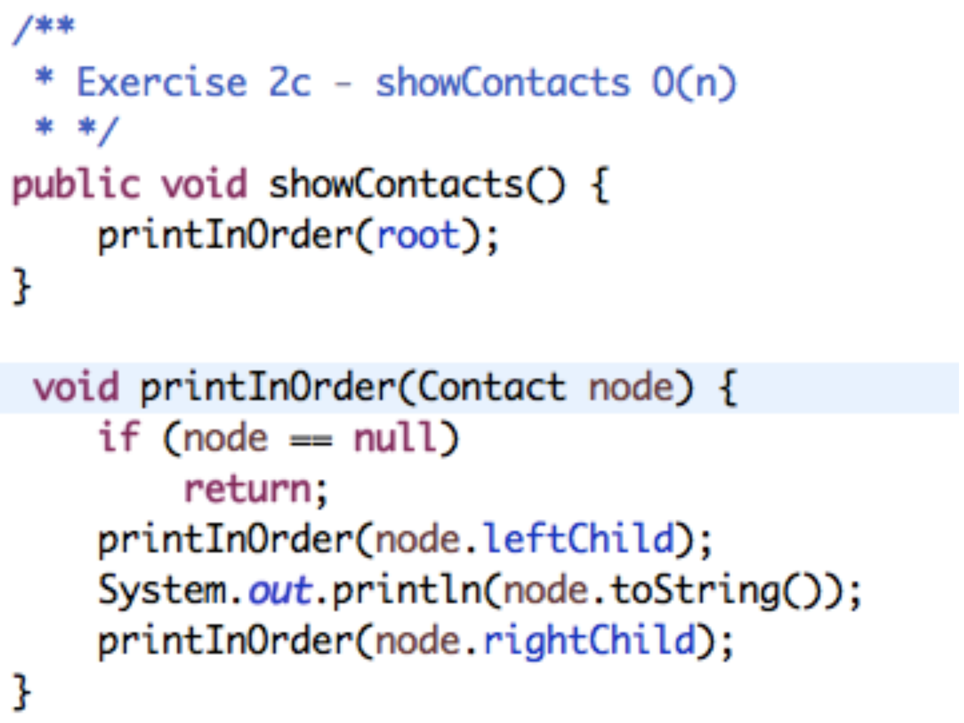


```java
public class AgencyBST {

    protected Contact root;
```

b) (1.25 points) Implement a recursive method to store a contact.

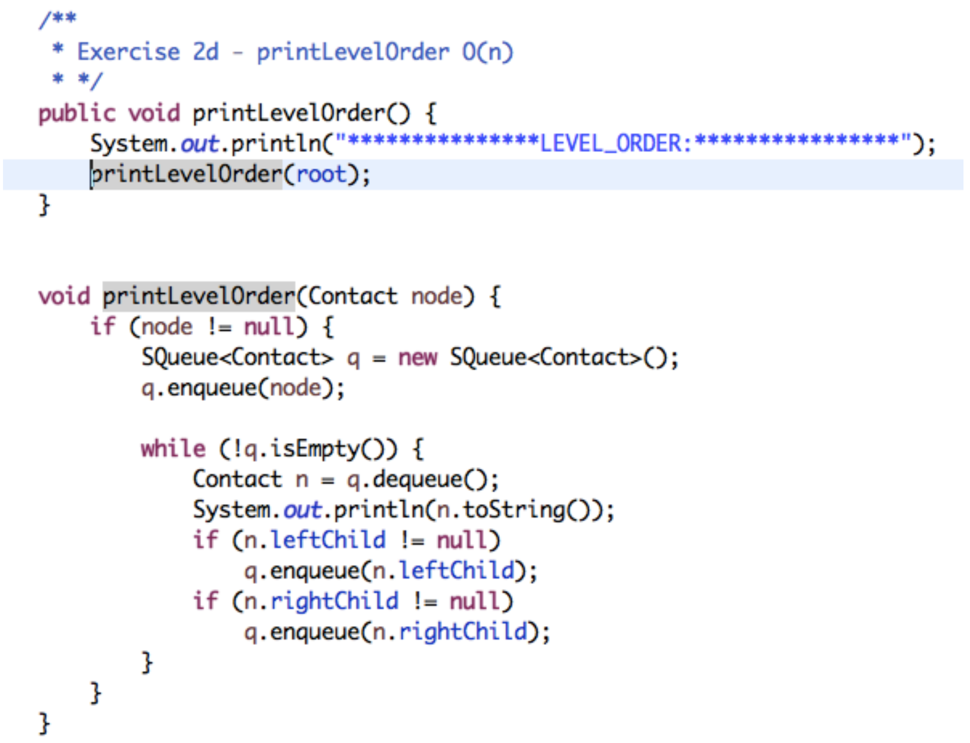
```java
public void insert(String name, String email) {
    Contact newNode = new Contact(name, email);
    if (root == null) {
        root = newNode;
        return;
    }
    insert(newNode, root);
}

private void insert(Contact newNode, Contact current) {
    String name = newNode.name;
    if (name.compareTo(current.name) == 0) {
        System.out.println("Cannot insert, the name already exists");
        return;
    }
    if (name.compareTo(current.name) < 0) {
        if (current.leftChild == null) {
            current.leftChild = newNode;
            newNode.parent = current;
            return;
        } else {
            insert(newNode, current.leftChild);
        }
    } else {
        if (current.rightChild == null) {
            current.rightChild = newNode;
            newNode.parent = current;
            return;
        } else {
            insert(newNode, current.rightChild);
        }
    }
}
```

c) (1.25 points) Implement a recursive method showing contacts in alphabetical order.



```
/**
 * Exercise 2c - showContacts O(n)
 * */
public void showContacts() {
    printInOrder(root);
}

void printInOrder(Contact node) {
    if (node == null)
        return;
    printInOrder(node.leftChild);
    System.out.println(node.toString());
    printInOrder(node.rightChild);
}
```

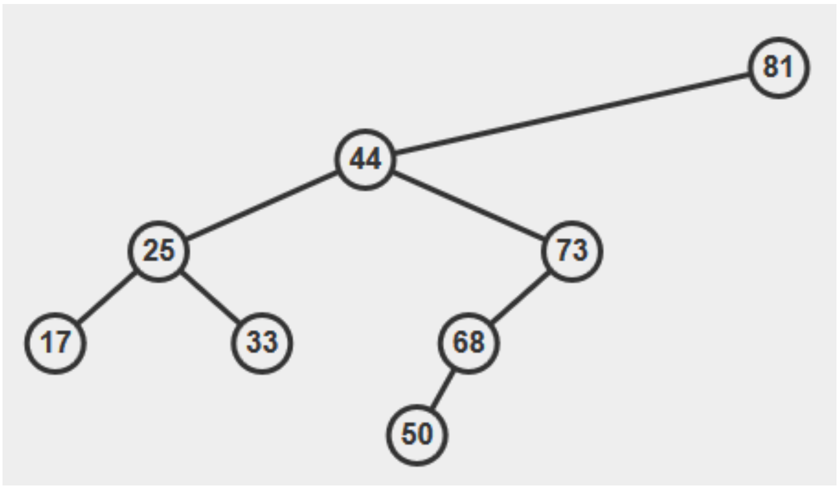
d) (2 points) Implement an iterative method to perform a level order traversal through the BST tree storing contacts. In this method you can use SQueue<E> class defined in EdaLib or any other class implementing a queue. You do not need to implement the code for SQueue<E> class.



```
/**
 * Exercise 2d - printLevelOrder O(n)
 * */
public void printLevelOrder() {
    System.out.println("***************LEVEL_ORDER:****************");
    printLevelOrder(root);
}


void printLevelOrder(Contact node) {
    if (node != null) {
        SQueue<Contact> q = new SQueue<Contact>();
        q.enqueue(node);

        while (!q.isEmpty()) {
            Contact n = q.dequeue();
            System.out.println(n.toString());
            if (n.leftChild != null)
                q.enqueue(n.leftChild);
            if (n.rightChild != null)
                q.enqueue(n.rightChild);
        }
    }
}
```

Problem 3 (1 point).

Given the following Binary Search Tree:



Please, answer the following:

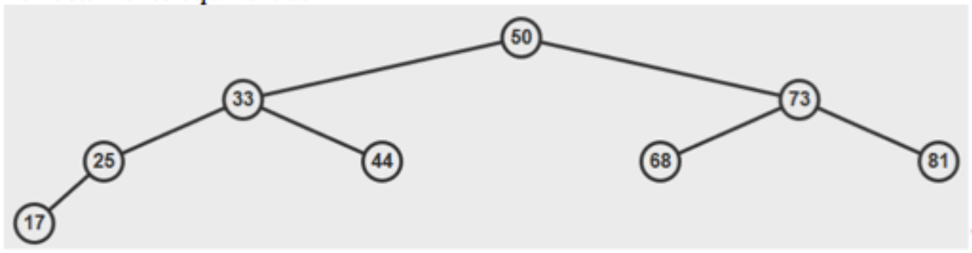
a) Size balanced version (0.50)

b) AVL version (height balanced) (0.50)

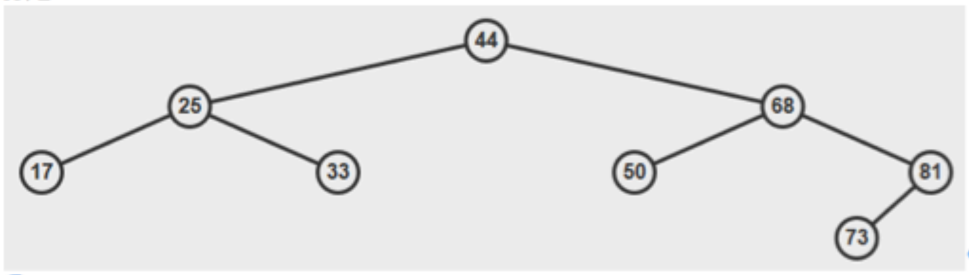
Note: The solution must include each of the steps followed to get the final result.

Solución:

Perfectamente equilibrado



AVL


44
25
68
17
33
50
81
73