NAME:
NIA:
GROUP:

Part B: Exercises Time: 45 minutes

Instructions:

- No books or other resources allowed.
- Do not forget to write your name, NIA and group in every answer sheet of paper.

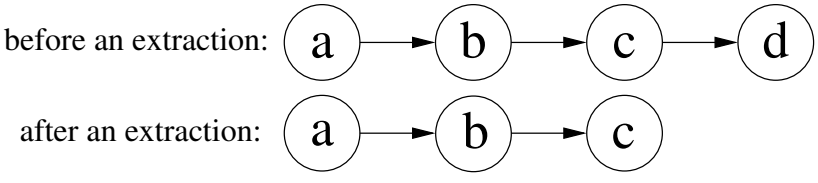
The following two Java classes represent a linked list:

```
public class Node {
    private Object info;
    private Node next;

    public Node(Object info, Node next) {
        this.info = info;
        this.next = next;
    }

    public Object getInfo() { return info; }
    public Node   getNext() { return next; }
    public void   setNext(Node next) { this.next = next; }
}
```

```
public class LinkedList {
    private Node first;
    public LinkedList() { first = null; }
    public void insert(Object o) { first = new Node(o, first); }
}
```

1 Extract the last element (2 points out of 5)

Add a "`public Object extract()`" method to the `LinkedList` class. It will extract the element at the end of the list and return its information. If the list is empty, it will return `null`. For instance, this is how an example list will look like before and after an extraction:



Do not use or create any other class or method than the ones mentioned so far.

```
    // extracts and returns the last element
    public Object extract() {
        if (first == null)
            return null;

        // find the last node and the next-to-last
        Node last = first;
        Node ntl = null;
        while (last.getNext() != null) {
            ntl = last;
            last = last.getNext();
        }

        // remove the last node and return it
        if (ntl == null)       // the list has only one element
            first = null;
        else                   // the list has more than one element
            ntl.setNext(null);
        return last.getInfo();
    }
```

2 Build a stack (2 points out of 5)

Fill in the dots (...) to create a stack based on the previous `Node` definition.

```
public class Stack {
    private Node first;
    public Stack() { first = null; }

    public void push(Object o) {
        // ...
    }

    // returns null if empty
    public Object pop() {
        // ...
    }
}
```

Do not use or create any other class or method than the ones mentioned so far.

```
    public void push(Object o) {
        first = new Node(o, first);
    }

    // returns null if empty
    public Object pop() {
        if (first == null)
            return null;
        Node tmp = first;
        first = first.getNext();
        return tmp.getInfo();
    }
```

3 Find a branch (1 points out of 5)

The following two Java classes represent a binary tree:

```
public class BNode {
    private String info;
    private BNode  left;
    private BNode  right;
}
```

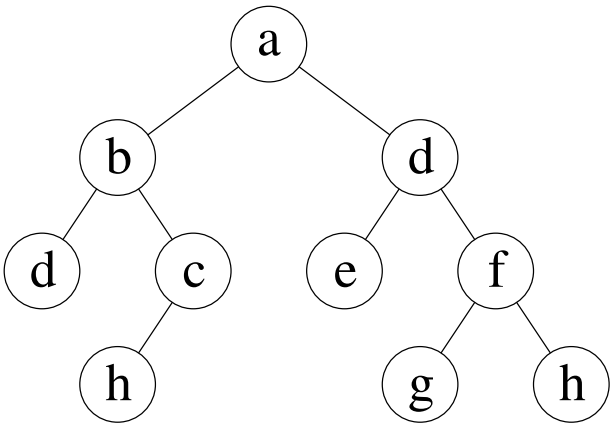
```
public class BTree {
    private BNode root;
    public BTree() { root = null; }

    public LinkedList findPath(String string) {
        LinkedList list = new LinkedList();
        if (root == null)
            return list;

        // push the path into an stack
        Stack stack = new Stack();
        root.findPath(string, stack);

        // pop the path to the list and return it
        String s;
        while ((s = (String) stack.pop()) != null)
            list.insert(s);
        return list;
}
```

The "`LinkedList findPath(String string)`" method returns the path between the root and a node that has `string` as its info. Paths are represented as a linked list containing the strings of all the nodes in the path.

Add the method "`public boolean findPath(String string, Stack stack)`" to the class `BNode`, so that the path returned by "`LinkedList findPath(String string)`" is the shortest pre-order path.

Do not use or create any other class or method than the ones mentioned so far.

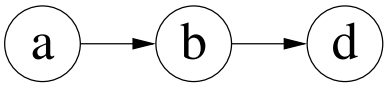
As an example, given the following tree:



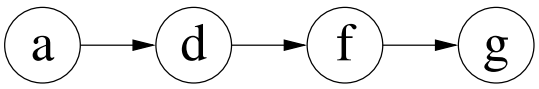
The results of calling "`findPaths(String string)`" with "a", "d", "g" and "h" as its arguments will be the following lists:

findPath("a"):

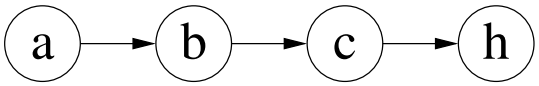
findPath("d"):



findPath("g"):



findPath("h"):




```
public boolean findPath(String string, Stack stack) {
    stack.push(this.info);
    if (string.equals(this.info))
        return true;
    if (left != null && left.findPath(string, stack))
        return true;
    if (right != null && right.findPath(string, stack))
        return true;
    stack.pop();
    return false;
}
```