***Penguin register***

Recently, the penguin colony's population has increased significantly. It's getting harder and harder to find a particular penguin by name. The colony therefore wants us to develop an efficient way to manage all penguins by name.

Each penguin has a unique name of any length, made up of the capital letters from 'A' to 'Z'. There are no umlauts in the names. These characters are referred to below as the "alphabet".

In order to efficiently manage the penguins based on their names, we will construct a tree that stores the names of the penguins. Each node of the tree can be assigned to a character string above the alphabet. This penguin is saved as an object at every node of the tree where the assigned character string corresponds to the name of a penguin.

The assignment of a node in the tree to a character string takes place via its position in the tree. It works like this:

The empty word (i.e. the character string of length 0) is assigned to the root.

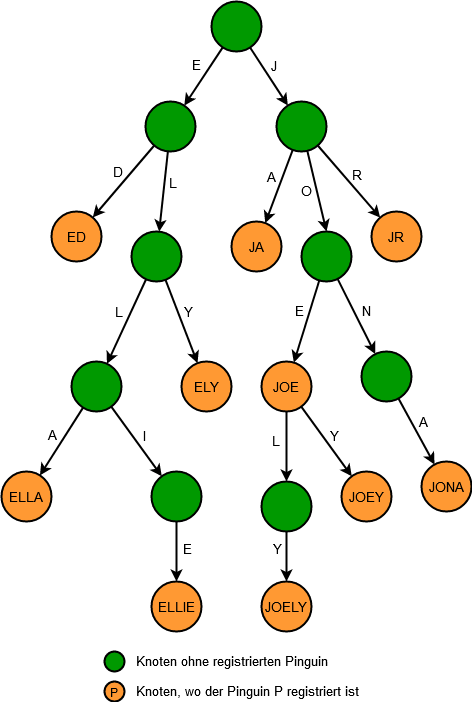
Each outgoing edge of a node to a successor stands for a character from our alphabet.

A node x with the predecessor v in the tree is also assigned to a unique character string. This string is the concatenation of the string from v and the sign of the edge from v to x.

Thus we can assign a node x to a unique character string by memorizing the path (a sequence of edges) from the root node to x.

If a penguin is to be registered with a name, a path is created in the tree so that the character string assigned to the end node is the name of the penguin, if such a path does not yet exist. After making sure that a node matching its name exists, the penguin object is saved in this node.

In the tree in the diagram on the bottom, the 10 penguins with the names ED, ELLA, ELLIE, ELY, JA, JOE, JOELY, JOEY, JONA and JR are registered.



***The registry***

In this task we expand the basic structure of the given class PenguinRegister. An object of this class corresponds to a node in the tree.

Create only the following attributes:

private penguin penguin, the registered penguin at this node. This attribute has the value zero if no penguin is registered at this node.

private PenguinRegister [] children, a field of length 26, where field element 0 corresponds to edge A, element 1 to edge B, and so on, up to element 25, which corresponds to edge Z.

If an edge does not exist, the field element has the value zero.

Implement the following public methods with the assumption that all method calls pass valid parameters, taking care to exploit the structure of the tree and only look at the parts of the tree that are necessary.

PenguinRegister put (String name, Penguin penguin) iteratively searches for the node that belongs to name and registers the given penguin object there and returns the node.

If the node does not yet exist, this method will create it.

Penguin get (String name) iteratively searches for the registered Penguin object with the name and returns it. If no penguin with the name exists, zero is returned.

boolean contains (String name) returns true if a penguin is registered with the given name, also implemented iteratively.

int size () recursively counts how many penguins are already registered in the tree.

String findName (Penguin penguin) recursively searches for the given penguin object and reconstructs the name that is returned. If the object is not registered, then null is returned.

Penguin remove (String name) recursively searches for a Penguin that was registered with the name. If found, it is removed from the tree and returned. Otherwise null is returned.

According to this method, the tree should still have the minimum size, i.e. H. there is no subtree whose size () is 0.

EntryList getAll () returns an EntryList listing all registered penguins in ascending order by name. An EntryList object is simply a list of Entry objects, which in turn is a pair of the name and the Penguin object of a registered penguin.

This method should also be implemented recursively.

EntryList getAllWithPrefix (String prefix) returns an EntryList, which lists all registered penguins in ascending order by name, whose names start with the prefix.

This method should also be implemented recursively and may only search the subtree that belongs to prefix.

EntryList getAllMatching (String pattern) returns an EntryList that lists all registered penguins in ascending order by name, whose names match the given pattern. Pattern is an expression that can only contain literals from the alphabet and periods ('.'). A '.' in the pattern it means that any character of the alphabet can appear in a matching character string at this point. The character strings matched by the pattern are just as long as the pattern. For example, JOEY and JONA match the pattern .O .., but ELLA, JOE and JOELY do not. This method should again be implemented recursively and may only visit the subtrees that would actually come into question according to the pattern.

You may implement any other private helper methods, but no further attributes (except static constants). EntryList objects may no longer be sorted after they have been created.

Input example:

PenguinRegister register = new PenguinRegister();

Penguin ella, joe, jona;

PenguinRegister joeSubtree;

register.put("ED", new Penguin("ED is chilling"));

register.put("ELLA", ella = new Penguin("ELLA wishes you merry christmas"));

register.put("JR", new Penguin("JR as in junior"));

joeSubtree = register.put("JOE", joe = new Penguin("JOE loves frozen fish"));

register.put("JOEY", new Penguin("JOEY to the world ♬"));

register.put("ELLIE", new Penguin("ELLIE loves her sister"));

register.put("JOELY", new Penguin("JOELY eats some jelly"));

register.put("ELY", new Penguin("ELY often gets mistook for her sister"));

register.put("JA", new Penguin("JA sir"));

register.put("JONA", new Penguin("JONA listens to music"));

System.out.println(register.get("ELLA"));

System.out.println(register.get("JOEY"));

System.out.println(register.get("CAROL"));

System.out.println(register.contains("ELLA"));

System.out.println(register.contains("CAROL"));

System.out.println(register.size());

System.out.println(register.findName(ella));

System.out.println(register.findName(joe));

System.out.println(register.findName(new Penguin("JOE loves frozen fish"))); //impostor!

jona = register.remove("JONA");

System.out.println(ella == register.remove("ELLA") ? "ELLA has been removed" : "not removed?");

System.out.println(register.size());

register.put("ELLA", ella);

register.put("JONA", jona);

System.out.println(register.getAll());

System.out.println(joeSubtree.getAll());

System.out.println(register.getAllWithPrefix("JOE"));

System.out.println(register.getAllWithPrefix("E"));

System.out.println(register.getAllMatching(".O.."));

Output Example:

ELLA wishes you merry christmas

JOEY to the world ♬

null

true

false

10

ELLA

JOE

null

ELLA has been removed

8

["ED": "ED is chilling", "ELLA": "ELLA wishes you merry christmas", "ELLIE": "ELLIE loves her sister", "ELY": "ELY often gets mistook for her sister", "JA": "JA sir", "JOE": "JOE loves frozen fish", "JOELY": "JOELY eats some jelly", "JOEY": "JOEY to the world ♬", "JONA": "JONA listens to music", "JR": "JR as in junior"]

["": "JOE loves frozen fish", "LY": "JOELY eats some jelly", "Y": "JOEY to the world ♬"]

["JOE": "JOE loves frozen fish", "JOELY": "JOELY eats some jelly", "JOEY": "JOEY to the world ♬"]

["ED": "ED is chilling", "ELLA": "ELLA wishes you merry christmas", "ELLIE": "ELLIE loves her sister", "ELY": "ELY often gets mistook for her sister"]

["JOEY": "JOEY to the world ♬", "JONA": "JONA listens to music"]