**Implementing a BinaryTree Class**

2 ways:

* Keeping a node and 2 references for the children
* Keeping an array
  + 1 element for the 1st level, 2 elements for the 2nd level, 4 elements for the 3rd level etc.

Implementing With Nodes and Links

Diagram, schematic

Description automatically generatedJust as for a linked list, a node consists of a data part and links to successor nodes

* The data part is a reference to type E
* A binary tree node must have links to both its left and right subtrees

Node <E> is declared as an inner class withing BinaryTree<E>.

Text, letter

Description automatically generatedStatic is because Node class doesn’t need to access the outer class.

We defined Node class as protected but we have defined it private in linked list. That’s because if we extend BinaryTree class with another binary tree class like BinarySearchTree, this node structure has to be accessed by the subclasses that extends BinaryTree class.

Assume the tree is referenced by variable bT (type BinaryTree):

Diagram

Description automatically generated

Table

Description automatically generated



Class heading and data field declarations:

Text, letter

Description automatically generated

Serializable means class instances can be written into a file directly. It copies into file with deep copy.

The Serializable interface defines no methods. It provides a marker for classes that can be written to a binary file using the ObjectOutputStream and read using the ObjectInputStream

Using ObjectOutputStream and ObjectInputStream

The Java API includes the class ObjectOutputStream that will write to an external file any object that is declared to be Serializable

To declare an object Serializable, add “***implements Serializable***” to the class declaration.

The Serializable interface contains no methods, but it serves to mark the class and gives you control over whether or not you want your object written to an external file.

To write a Serializable object to a file:

Graphical user interface, text, application

Description automatically generated

A deep copy of all the nodes of the binary tree will be written to a file.

To read a Serializable object from a file:

Graphical user interface, text, application, email

Description automatically generated

Do not recompile the Java source file for a class after an object of that class has been serialized.

Even if you didn’t make any changes to the class, the resulting .class file associated with the serialized object will have a different class signature

When you attempt to read the object, the class signatures will not match, and you will get an exception

- Constructors

The no-parameter constructor: The constructor that creates a tree with a given node at the root:

Text

Description automatically generated Text

Description automatically generated with medium confidence

The constructor that builds a tree from a data value and two trees:

Text, letter

Description automatically generated

- getLeftSubtree and getRightSubtree

Text

Description automatically generated

If we want to return an empty binary tree instead of null binary tree, we can write “***return new BinaryTree<E>(null);***” to else part.

getRightSubtree method is symmetric.

- isLeaf

A picture containing graphical user interface

Description automatically generated

- toString

Generates a string representing a preorder traversal in which each local root is indented a distance proportional to its depth

Text

Description automatically generated with medium confidence

Last two parameters in the preOrderTraverse (1 and sb) are not necessary. They are for efficiency and to make the output better.

sb is StringBuilder and it is for improving the performance of the toString method.

1 is for make the string a little bit more readable, it is level value.

- preOrderTraverse

Text, letter

Description automatically generated

Diagram

Description automatically generated with low confidence

Running time of toString() and preOrderTraverse():

Running time of sb.toString() depends on length of the string.

Running time of preOrderTraverse() depends on number of nodes. If number of nodes is n, running time would be (n) (from 2n+1 as we mentioned in preOrderTraverse part) except indentation (sb.append(node.toString() is taught as constant.).

We have to add depth of each node to find the number of indentations. Depth for a single node is linear in the worst case, constant in the best case. Addition of number of depth of each node is () for the worst case. If tree is a perfect tree (or if it cannot be perfect, then complete tree), it is best case. Total depth if tree is perfect : (nlogn) *{actually, is it not but O?}*(height is logn, there are n nodes). We said O because only lower nodes has logn depth, other ones have lower depths.

In perfect tree, there are almost n/2 nodes at the lowest level and height is logn. So if you add depth of lowest nodes only, it is (n/2)logn and total height is ((n/2)logn).

With(nlogn) and ((n/2)logn) , we can say addition of number of depth of each node is for the best case.

SO, FOR INDENTATION:



* worst case ------> ()
* best case ------>

So total time of preOrderTraverse depends on for statement, not depends on if and else part.

- Reading a Binary Tree / readBinaryTree

If we use a Scanner to read the individual lines created by the toString and preOrderTraverse methods, we can reconstruct the tree

Graphical user interface, text, application

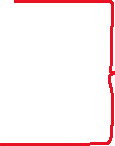
Description automatically generated

Text, letter

Description automatically generated

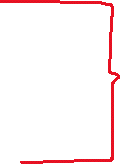
Diagram

Description automatically generated with low confidence ROOT



LEFT SUBTREE

RIGHT SUBTREE



Performance depends on number of nodes. Also we have empty spaces that has to be skipped. Size of all these indentations are more than the number of nodes (it is quadratic to the number of nodes). Scanner has to read every character in the file including spaces.

If you don’t consider indentations, performance is linear. Since we process each node at each recursive call. Number of calls is 2n+1 for n nodes (n: number of nodes, n + 1: number of null nodes).