Okeoghene Excel Omorobe

CMSC 204

Assignment 5 Documentation

**Learning Experience**

Through this assignment, I gained more insight into generic classes and recursive data structures and how binary trees can be used to solve problems in real life such as decoding Morse code. Creating a MorseCodeTree and implementing it with an underlying linked binary tree structure forced me to think carefully about traversal patterns and node placement. Every letter of the alphabet had to be positioned in its proper location in the tree according to its corresponding Morse code pattern and this made me better understand how to traverse trees. For example, how dots (.) translate to left movements and dashes (-) translate to right movements.

The biggest hurdle I had to overcome was making sure the tree was built properly so that the in-order traversal print matched the expectations set by the test cases. This made me double-check both my insertion logic and how I thought the structure of the tree would influence the print outcome. Debugging the printTree() test was an enlightening experience because it illustrated how small differences in build order or structure could radically alter the outcome when the logic is correct. It was also an excellent reminder of the need to maintain consistency between expected and realized outcomes in unit testing.

Aside from the technical know-how, this project has educated me on how to design utility classes and effectively utilize static methods for centralized operations and functions like parsing files and string manipulation. The need to create JUnit test cases solidified my test-driven development practice by reemphasizing the need to write tests to capture not just typical use scenarios but edge cases such as an empty input. The experience will be useful in developing more sophisticated applications with more advanced data structures and algorithmic problems in the future.

**Pseudocode**

* **Tree Node<T> - Generic Tree Node**

Class TreeNode<T>

Variables:

data: T

left: TreeNode<T>

right: TreeNode<T>

Constructor(data: T)

this.data = data

this.left = null

this.right = null

Constructor(data: T, left: TreeNode<T>, right: TreeNode<T>)

this.data = data

this.left = left

this.right = right

Method getData()

return data

Method setData(data: T)

this.data = data

Method getLeft()

return left

Method setLeft(left: TreeNode<T>)

this.left = left

Method getRight()

return right

Method setRight(right: TreeNode<T>)

this.right = right

* **MorseCodeTree – Builds and Traverses the tree**

Class MorseCodeTree implements LinkedConverterTreeInterface<String>

Variable:

root: TreeNode<String>

Constructor()

Call buildTree()

Method getRoot()

return root

Method setRoot(newNode: TreeNode<String>)

root = newNode

Method buildTree()

Create root with empty string

Insert all Morse code letter mappings:

insert(".", "e")

insert("-", "t")

insert("..", "i")

...

(continue for all 26 letters)

Method insert(code: String, letter: String)

Call addNode(root, code, letter)

Method addNode(current: TreeNode<String>, code: String, letter: String)

if code length == 1:

if code == ".":

set left child of current to new TreeNode(letter)

else if code == "-":

set right child of current to new TreeNode(letter)

else:

first = code[0]

rest = code[1:]

if first == ".":

if left child is null:

set left to new empty TreeNode

recurse addNode(left child, rest, letter)

else if first == "-":

if right child is null:

set right to new empty TreeNode

recurse addNode(right child, rest, letter)

Method fetch(code: String)

return fetchNode(root, code)

Method fetchNode(current: TreeNode<String>, code: String)

if code length == 0:

return current.data

if code[0] == ".":

return fetchNode(current.left, code.substring(1))

else:

return fetchNode(current.right, code.substring(1))

Method toArrayList()

list = new empty ArrayList

Call LNRoutputTraversal(root, list)

return list

Method LNRoutputTraversal(current: TreeNode<String>, list: ArrayList<String>)

if current is not null:

recurse on current.left

add current.data to list

recurse on current.right

Method delete(data)

throw UnsupportedOperationException

Method update()

throw UnsupportedOperationException

* **MorseCodeConverter – Utility Class**

Class MorseCodeConverter

Variable:

static tree: MorseCodeTree = new MorseCodeTree()

Static Method convertToEnglish(code: String)

Split input on " / " to get words

For each word:

Split word on " " to get letters

For each letter:

use tree.fetch(letter) to get character

Concatenate characters to form word

Join words with space and return final string

Static Method convertToEnglish(file: File)

Read file contents as single string

Call convertToEnglish(string)

Static Method printTree()

Call tree.toArrayList() and return joined string