|  |
| --- |
| Project 3B  Erik Sklocic  Joseph Jennings  Sonal Sinha |

Design

**Data Structures Used:**

**Binary tree-** The main structure used in this project is a binary tree. With a depth of 4, the binary tree we used stored every letter of the alphabet with the corresponding morse code. Each piece of data was stored in a node. The tree was set up so that going left to right would access the keys with dots (“.”) and going right to left would access the keys with dashes (“-”). Going further right or left would add more dashes/dots, making up the morse codes for specific letters.

**ArrayList-** An arraylist is used many times throughout the code, specifically in the main() method and the buildTreeNode() method. In the main() method, an arraylist named inputArray and array name tempArr are created to store the information from the Morse\_Code.txt file. Information is first stored in tempArr and then added into inputArray. Then, the buildTreeNode() method is called and inputArray is sent as a parameter. In the buildTreeNode() method, an arraylist called lists is created and sets it equal to inputArray. In this method, both inputArray and tempArr are manipulated to access different parts of the tree.

**Array-** Similar to arraylists, arrays are also used in our code. As stated above, an array is created in the main() method and referenced many times in other parts of the code, like in the buildTreeNode() method. We chose to use Arraylists and arrays because we figured it would be the easiest thing to store data from the tree.

**Structure**

**Main.java :** The main.java class file is the file that contains all of our methods, and main functions of our code.

**main() method:** Gets the input from the user, stores it in a variable and passes it to be decoded and encoded, depending. The method also utilizes two different scanner objects, “scanner” and “scnr”. Each scanner object has a different function. The “Scanner” scanner object is used to ask the user for an input and stores the input in a string. The second is used when building the binary tree to receive the input from the input file and then this is stored in a variable which is then passed to buildTreeNode()

**buildTreeNode() method:** Builds a tree node based on the list it was given and returns a root node with a fully built binary tree.

**encode() method:** Receives the root of the tree and the user input string and iterates through the string letter by letter. It then calls the encodeLetter() method to perform the last operations for encoding the string.

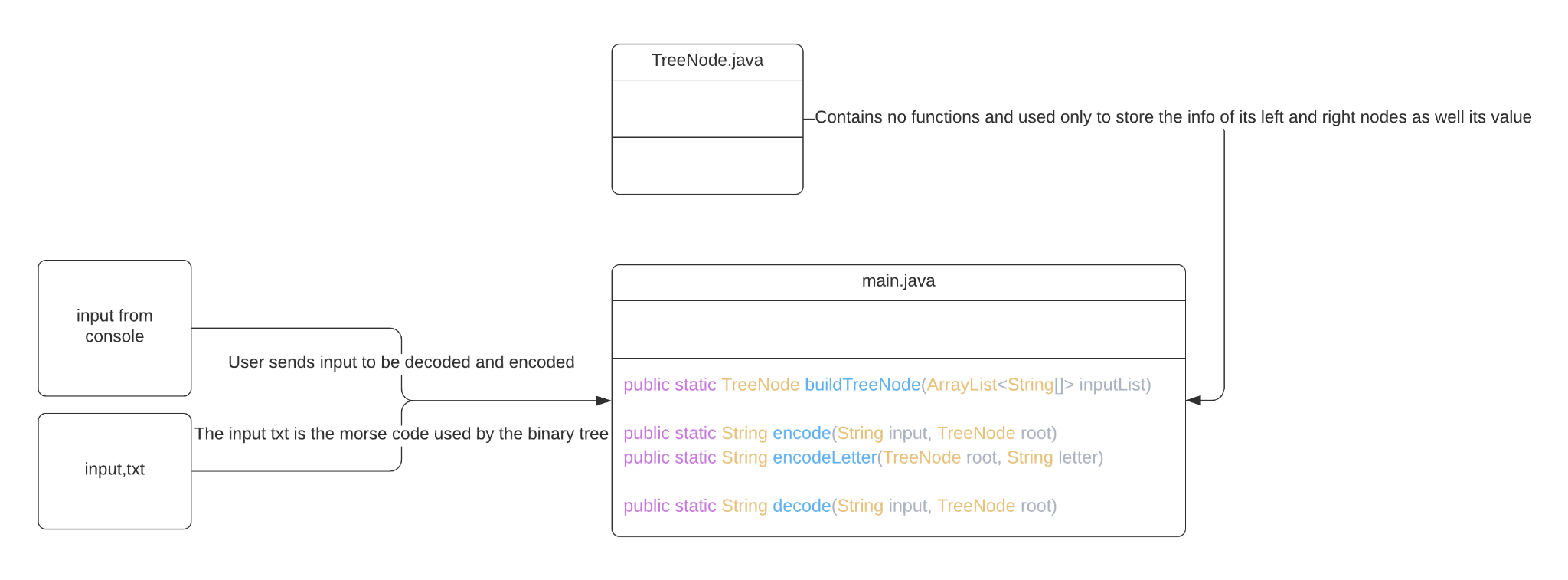
**encodeLetter() method:** Receives a string from the previous method and the root of the tree and encodes it from English to Morse Code.

**decode() method:** Decodes the message from Morse Code to English by splitting the encoded string by the spaces (each MorseCode letter is differentiated by a single space while each word is differentiated by a double space) and stores it in a new array called inMsg. It then uses a for loop to iterate through inMsg and checks whether or not there is a space. If a space string exists, a space will be added to the final decoded message, if not, it checks whether the character is a dash or dot. Depending on the character, it will either go to the right or left branch of the tree in order to determine the letter that corresponds to the Morse Code. Once found, the letter is added to the final decoded string (decodeMsg) and concatenated so that the letters are all connected in the same string, then repeats the process for the other letters in the code.

**Class TreeNode :** Keeps track of left and right nodes as well its value.

**Morse\_Code.txt:** Contains the morse code and the letter that represents it

UML



Assignments

Erik Sklocic: Assigned to making the binary tree and github

Joseph Jennings: Assigned to making the decoder

Sonal Sinha: Assigned to making the encoder

Test Case

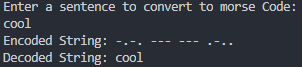
Test Case 1:

Expected:

Input: cool

Output: -.-. --- --- .-.. ,cool

Result:



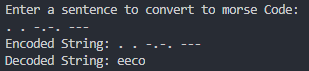
Test Case 2:

Expected:

Input: . . -.-. ---

Output: . . -.-. --- ,eeco

Result:



Improvements

The improvements for this project aren’t very abundant compared to our previous project. The improvements that could have been made are mainly our efficiency in our methods. For example the way the binary tree that is built could be used for encoding and decoding letters and messages could have been improved to not contain as many loops as it has now. The decode() method has a time complexity of O() which could have been reduced by not splitting the string, therefore, not needing to iterate through the initial list. Also, instead of having one big main.java file containing all the methods, we could have split up the methods and used classes instead, which would leave us with a more organized and readable code.