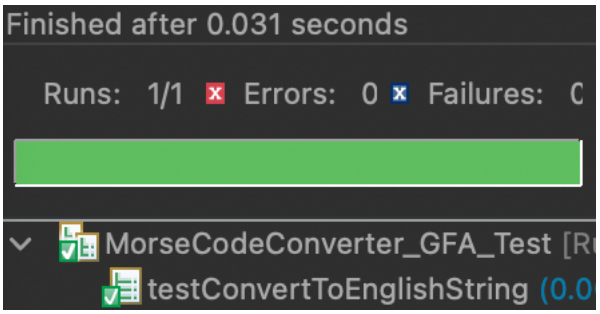
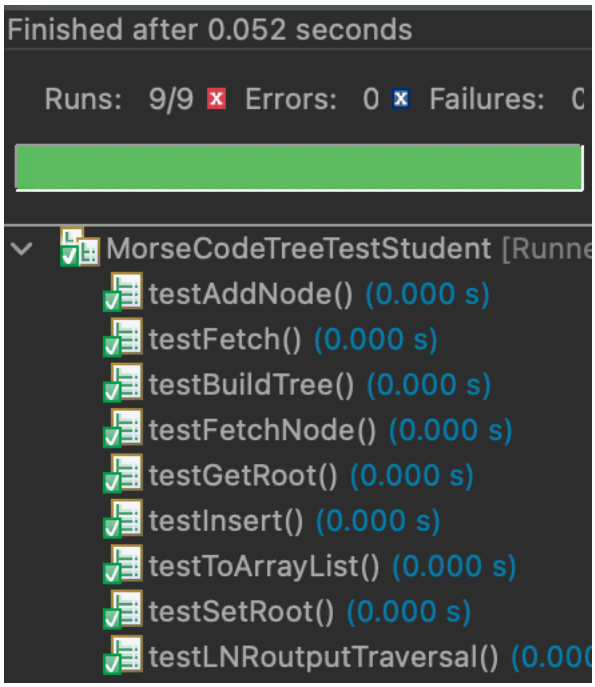
Project 5 - Bushman
Design and Algorithm

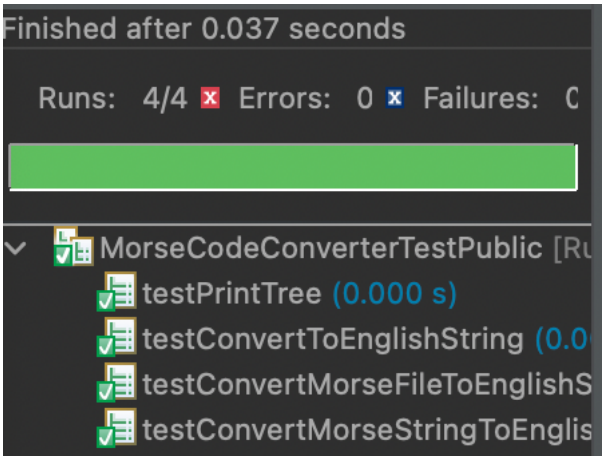
I first looked at the three class we had to create and figured that the TreeNode class would be the easiest and most straight forward. I created the necessary data nodes, along with the left and right child. I created a constructor that took in a given node and assigned it to my data and set the left and right child to null. I then created another constructor to deep copy based on the given node. Lastly, I returned the data in the getData() method. I then moved onto the MorseCodeTree class. I first created my root variable for the class. I created the basic constructor that calls the buildTree method as instructed. The getters and setters were straight forward. For the insert method, I first checked to see if the root was null. If it was, then I assigned the given letter to the class root. If it wasn't null, I then called on the recursive addNode method as instructed, including the given code and letter. For the addNode method, to cover all of my ends, I first checked if the code length was zero to start with. If it was, then I ended the method. If the code length was one, I then checked it was a dot. If it was a dot, then I set the root's left child to be the given letter. If it was a dash, I then set the root's right child to be the given letter. If the code length is greater than one, I first checked if the first character was a dot. If it was, I removed the first character for the given code. I then called the addNode method and made sure to use the root's left with the updated code and letter to traverse left. If the first character is a dash, I removed the first character for the given code. I then called the addNode method and made sure to use the root's right child with the updated code and letter to traverse right. For the fetch method, this was pretty straight forward. I just return and called on the fetchNode method with the class root and given code string. The fetchNode method followed the same layout as my addNode method. The only thing I did change was creating a string to hold the data corresponding of the given code. At the end of the method, I returned the a string of the letter based on the provided code. For delete and update, I just threw the UnsupportedOperationException method as instructed. For the buildTree method, I first just insert the root as insert("","""). After this, I inserted level by level the morse code along with the matching letter based on the description in our project document. For visuals sake, I made more sense to layers the code level by level so I could remember were I am actually traversing to. Once I read over the given interface, I realized I needed to change the first insert("","""") and actually use the setRoot method to set the root as "". Both seemed to do the same thing so I'm not sure if anything really changed here (any guidance here would be appreciated). For the toArrayList method, I first created my empty array list variable. I then called on the LNOutputTraversal method with the class root and empty array list. After the array list was filled, I then returned the filled array list. For the LNROutputTraversal method, I first checked if the root was null. If so, I ended the method. I then recursively called the LNROutputTraversal method and used the root's left child to traverse left. Once it was at the left most child and no other left children were left, I then added that root's data to the array list. I then recursively called the LNROutputTraversal method to traverse to the right child and so on and so forth until the traversal was complete. I then moved on to the MorseCodeConverter class. I first created my MorseCodeTree variable and created the empty MorseCodeConverter constructor based on the javadoc. I first worked on the printTree method. I created and filled an array list and used the MorseCodeTree variable to call on the toArraylist method. I created a empty string to hold the data. I used a for loop to run through the array list size. I then assigned each element in the array list to the string. After the for loop was over, I made sure to trim the string of any leading or trailing spaces and returned the completed string. I then moved on the String convertToEnglish method since file methods scare me. I created two string arrays to hold the words and letters and an empty string to hold the converted code in English. I used split method to split each of the words in the code into their own spot in the string array. I used a for loop to run through the length of all the words array. I then used the split method again to split each letter in the the current word by the space and assigned it to my letter array. I then used another for loop to run through the letters length. Since the fetch method is the only one that

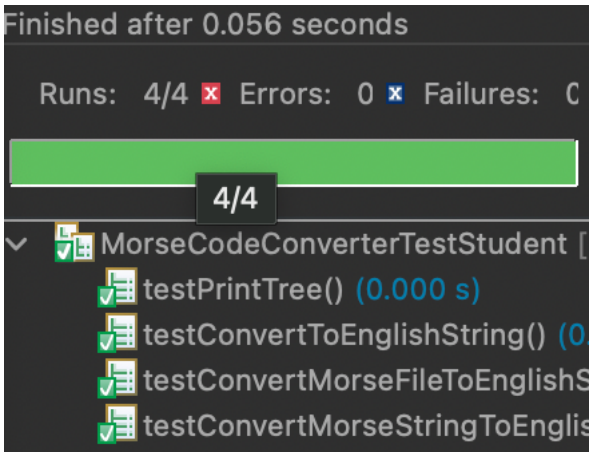
takes in a String and not fetchNode, I called on that method to retrieve the translated letter for each code. I added each one to my string holder. After the inner for loop was finished, I made sure to add a space between each finished word. After both loops finish, I used the trim method to remove any leading or trailing spaces and returned the code to English. Lastly, I worked not he convertToEnglish method for files. I'm not sure why I dreaded this one since I was the easiest file reader method have done. I first started with some variables for the Scanner of the given file along with a string to hold the converted code. I used an if statement to check if the file was empty. If it was, I returned an empty string just in case this was going to be one of the random instructor tests. I then used a while statement which occurs if the file has another line. If it does, I called on the convertedToEnglish method to read each line and convert. I assigned each converted line to the string holder. After the while loop finished, I then returned the completed converted string of the Morse code to English.

Junit Screenshots

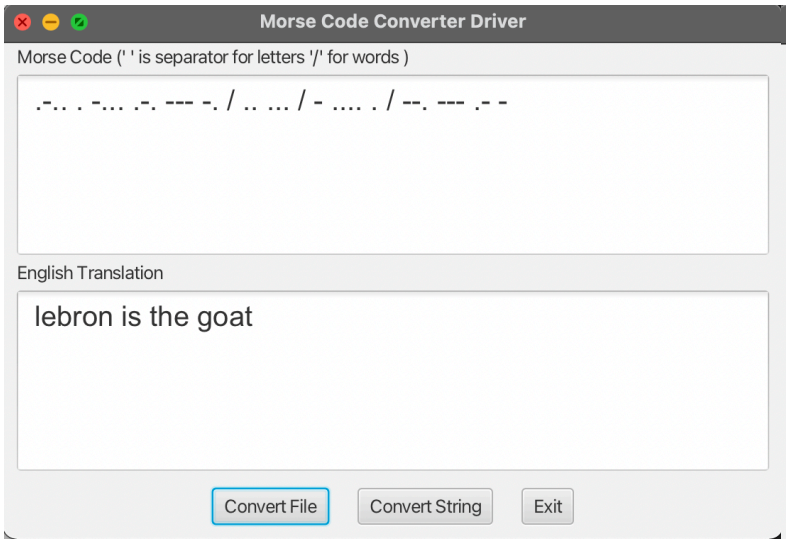


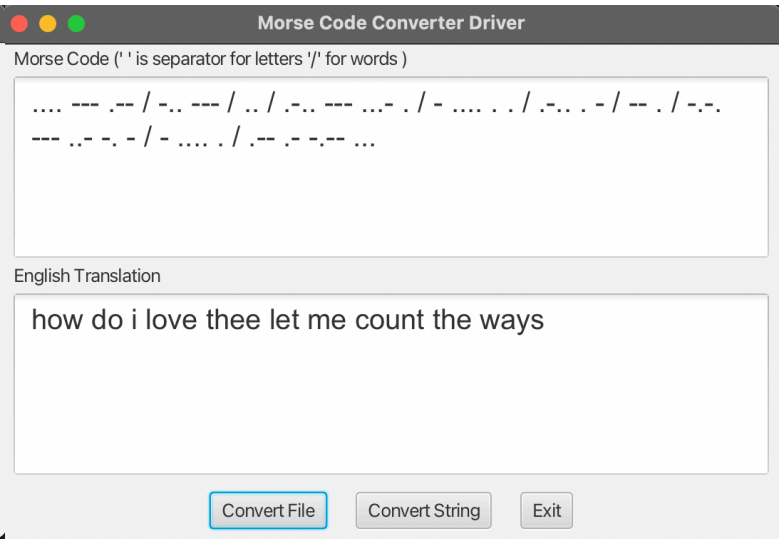


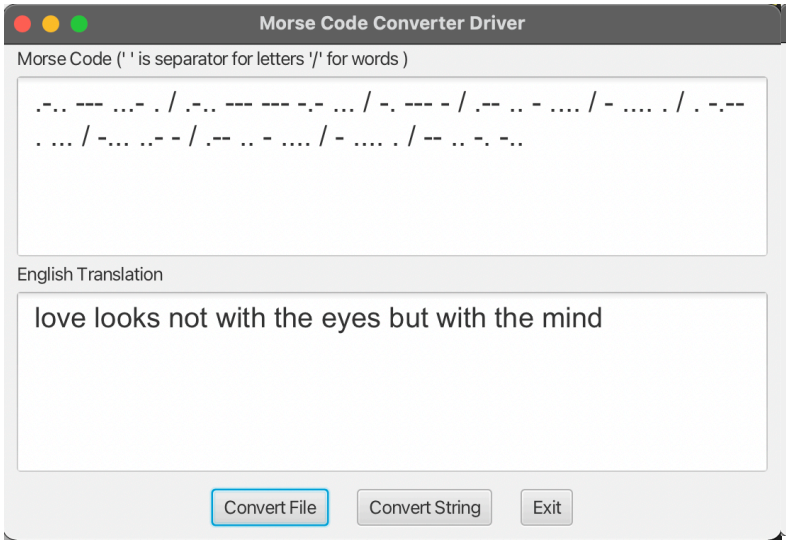


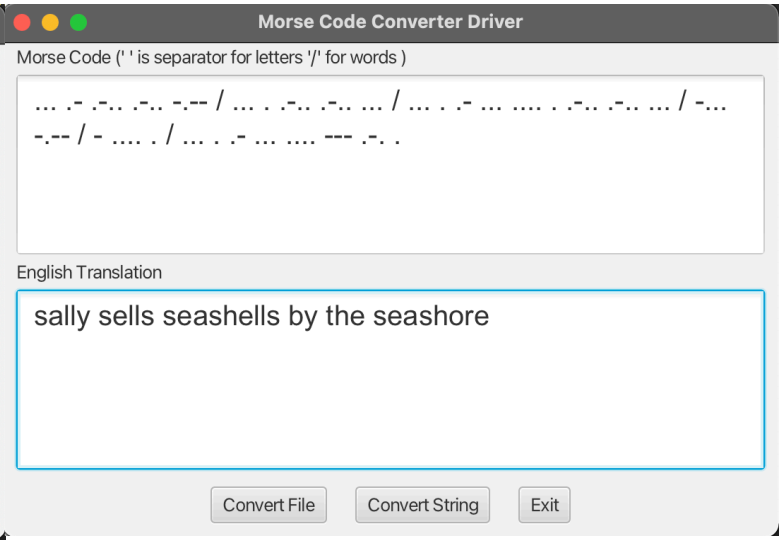


Application Screenshots



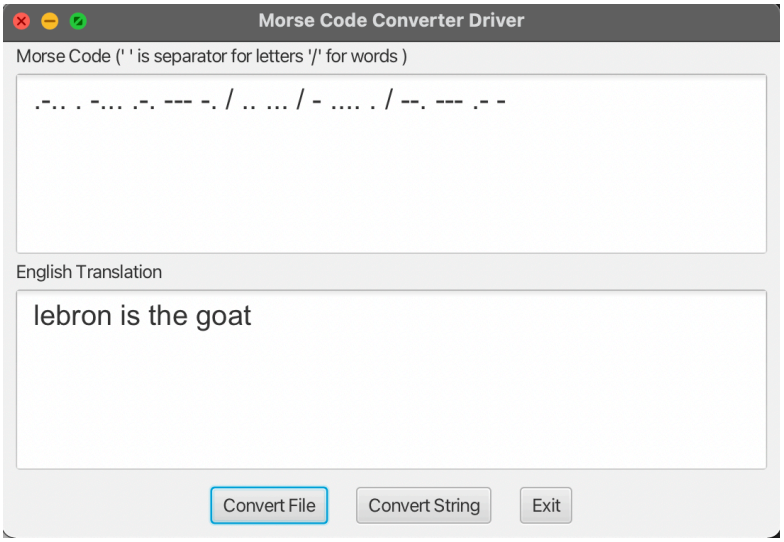




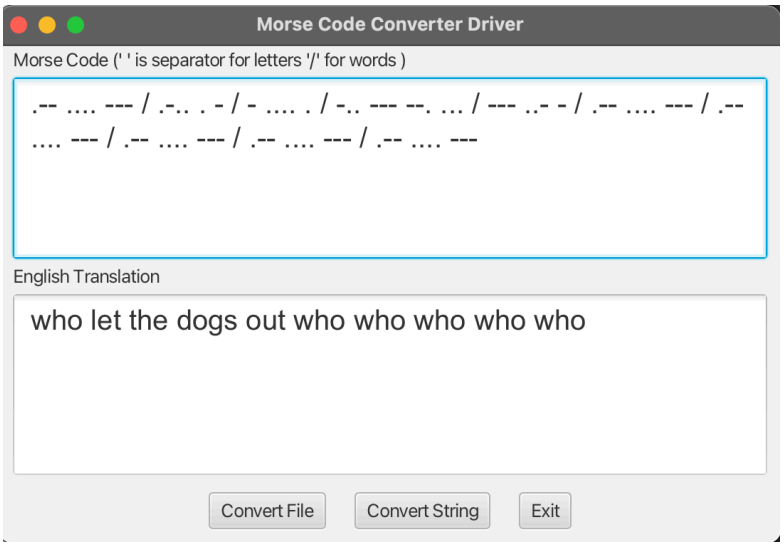


Test Screenshots

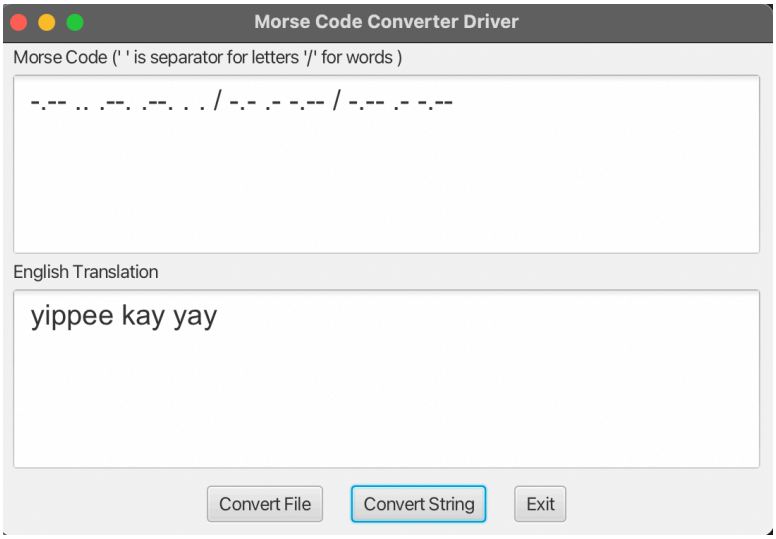
Test 1: testing the lebron.txt file
Expected: convert the code to say “lebron is the goat”
Reality: correctly converted the code to say “lebron is the goat”



Test 2: testing the MorseCode .-- .... --- / .-.. . - / - .... . / -.. --- --. ... / --- ..- - / .-- .... --- / .-- .... --- / .-- .... --- / .-- .... --- / .-- .... —
Expected: converts to say who let the dogs out who who who who who
Reality: correctly converted the code to say who let the dogs out who who who who who



Test 3: testing the MorseCode -.-- .. .--. .--. . . / -.- .- -.-- / -.-- .- -.--
Expected: converts to say yippee kay yay
Reality: correctly converted the code to say yippee kay yay



Learning experience:

I have learned a lot about building linked trees along with traversing them. I learned how to correctly add a node based on the given root, code and letter. I got comfortable traversing to the left child along with the right child. I got more comfortable creating Junit tests with no guide to follow other than our previous assigned assignments. I learned how to not just how to correctly write a tree into inorder traversal but also coding it as well. I learned how useful binary trees and traversal can be for Morse code. I would have never thought of using tree to traverse through the alphabet and generate a valid translation before this class. I got comfortable reading a file and calling a different method to convert each line that was read. I got more practice with using the split method for strings along with using the trim method. The trim method came in handy towards the end of methods since we needed to remove any left over spaces. The split method worked perfectly in creating arrays based on using the delimiter values. I learned to not always be afraid of file methods since this one was very straight forward and simple. I tend to put panic in myself when I see a method containing a file. I relearned that I could use substring to continuously remove the first character until I went through the whole code. This was helpful for the recursion methods that needed to go off the next character. Overall, I felt like I did pretty well on this assignment. I broke down the problem step by step and line by line. I feel like I made sure to capture any loose ends along with testing my code more throughly.

Some challenges I faced were creating a new Junit test class, dealing with recursion methods and remembering which line I am at when dealing with multiple recursions in one method. Creating the Junit tests was little tricky at first since we didn't have a guide or template to go off of. Dealing with recursions took some practice since I had to relearn that I could use substring to remove the first character and then call the recursions method again. Another thing was correctly using the left and right child during the recursions. Once I broke down everything and understood that we are using the child of the roots to traverse in either

direction, thing began making much more sense. Lastly, having two recursions in a method is not fun. The LNRoutputTraversal method was tricky for my head to wrap around. I realize that I need to use a pencil and paper to keep track of where I am in the code and which line I stopped on. It was very helpful once I created a little diagram of what method call I was on, which line I was stopped at and the current root I was at.

I don't really have any assumptions for this assignment. The Junits and other tests all ran without any modification needed. The only thing I could think of as an assumption is that we are assuming that the person entering in the code is using the correct symbols. If the person decided to an incorrect code letter with maybe an extra dot or dash in the wrong place, the program would crash. I feel that if someone were trying to decode an actual Morse code, that the Morse code would at least be correct in terms of using correct codes for the letters.

Question for Professor Thai: Is there an easier way to think of the LNRoutputTraversal method? Is it bad that I need to write down each method call, line and root to understand where I am at in the code? Is this something we should be able to figure out just in our heads or will this come to us better over time with more practice?