# Approach, design & algorithm

This project was extremely simple. I did not do much thorough planning to be quite honest. Before I started coding I took a look at specifications to get a general idea of the major components and how they “interlink” within the overarching project. These major components seemed very similar because the book implemented their binary tree in an analogous format. Therefore, no extra planning was truly necessary to successfully implement the “major links”.

I started with the most basic component first the TreeNode class. This class was quite straightforward there wasn’t anything too confusing. I was a little lost why we needed a copy constructor and some of the other methods the java docs specified but nonetheless, I implemented it and threw those worries aside until later. Spoiler alert, I still do not truly understand why some of the methods are specified other than for practice I’d assume. I did a little bit of console-based testing as I completed methods to ensure they worked as I intended then moved on to the MorseCodeTree class.

The MorseCodeTree class brought a bit more “challenge” than the prior class. I began by building the specified constructor which required building a chain of other methods. The length of this chain of methods, that is buildTree, insert, addNode, did not make complete sense to me but again I implemented it anyways in an effort to adhere to provided documentation. I then implemented the fetch method and a few other basic methods so that I could start doing some console testing. Once I was done adjusting my code I moved on to implement the other methods. I left input validation and error handling for later once I was done with the initial “working” version of the project. In the end, I came back to add logic to prevent the program from crashing due to faulty morse code strings. I tested these protections extensively within my test case section.

The MorseCodeConverter class was extremely simple to implement thanks to my use of the unmodified scanner object to iterate through tokens in the provided string. I purposefully chose not to change the delimiter and simply add an if statement to ignore the “/” character defining the end of a word. The method converting input from a file simply passed the one and only line in the file into the string converting method.

Within my JUnit test, I tried to test as many distinct characters as possible to triple check my tree was built properly.

# Assumptions

I do not think I made any major assumptions that would cause my project to deviate significantly from project specifications.

# Anything else to share

I have nothing else to share at this moment.

# Lessons Learned

**What did you learn?**

I don’t think we have been exposed to this type of binary search tree before. I am not entirely sure if this would necessarily be a traditional binary search tree because the left child isn’t “less than” the parent. However, we search in a very similar fashion by removing half the possibilities at each turn. I wonder what else could be implemented in this fashion.

**What issues did you encounter, if any?**

I had a few minor confusions which I touched on above but I did not encounter any major challenges.

**What would you have done differently?**

I wouldn’t change how I implemented the specifications for this project but I would like to revisit morse code processing in the future. Specifically, I think it would be more time efficient to use a hash map to define each character but it would also be easier from a logical implementation perspective. Also, it would be interesting to expand support for additional characters. However, passed that I do not see any way to meaningfully expand on this project.

**How can you apply this concept in the future?**

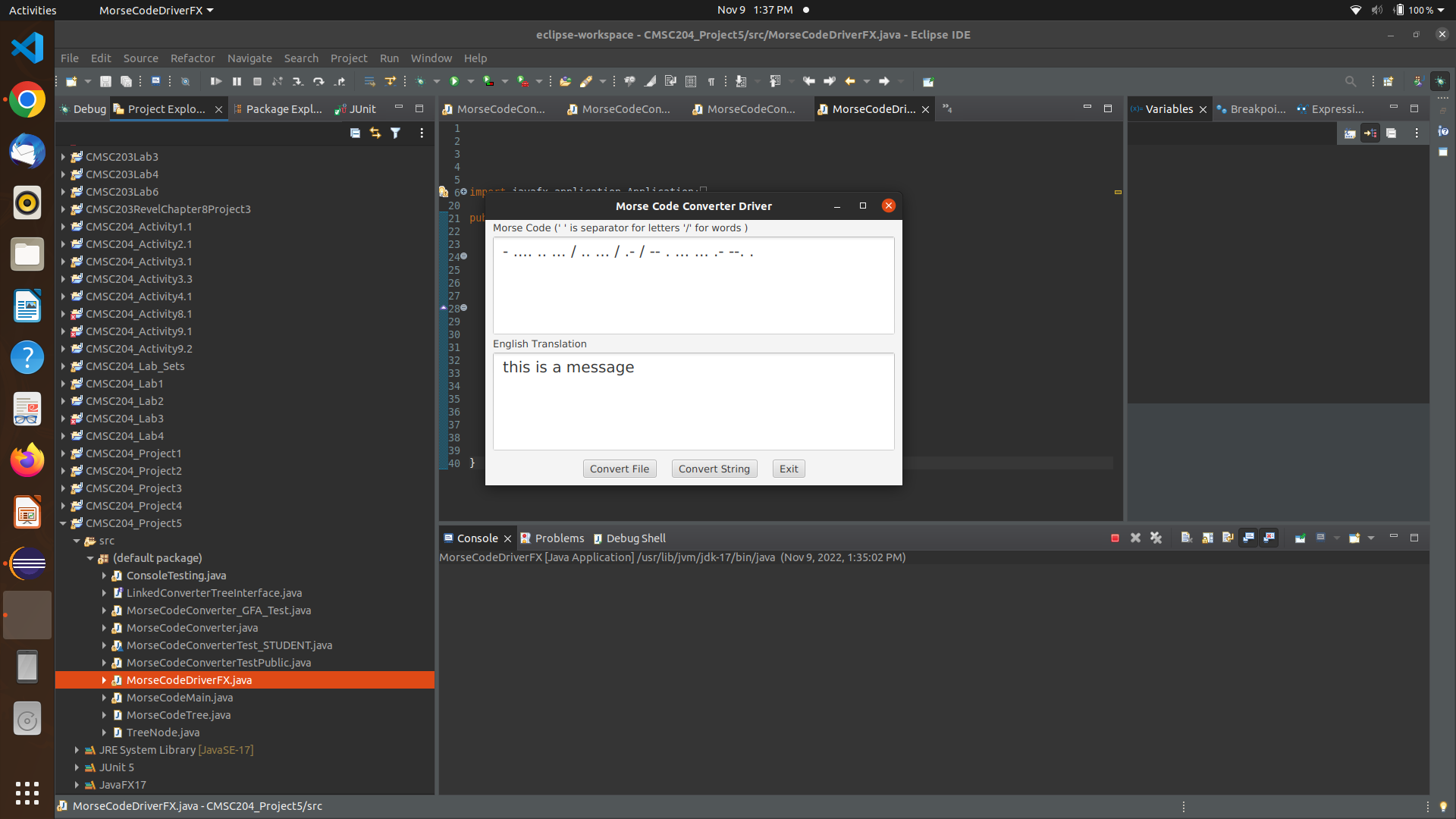
I believe binary trees will be a useful concept in the future. However, it is hard to imagine a scenario in which I need to create a binary tree and traverse it based on a set of very specific instructions as we did here.

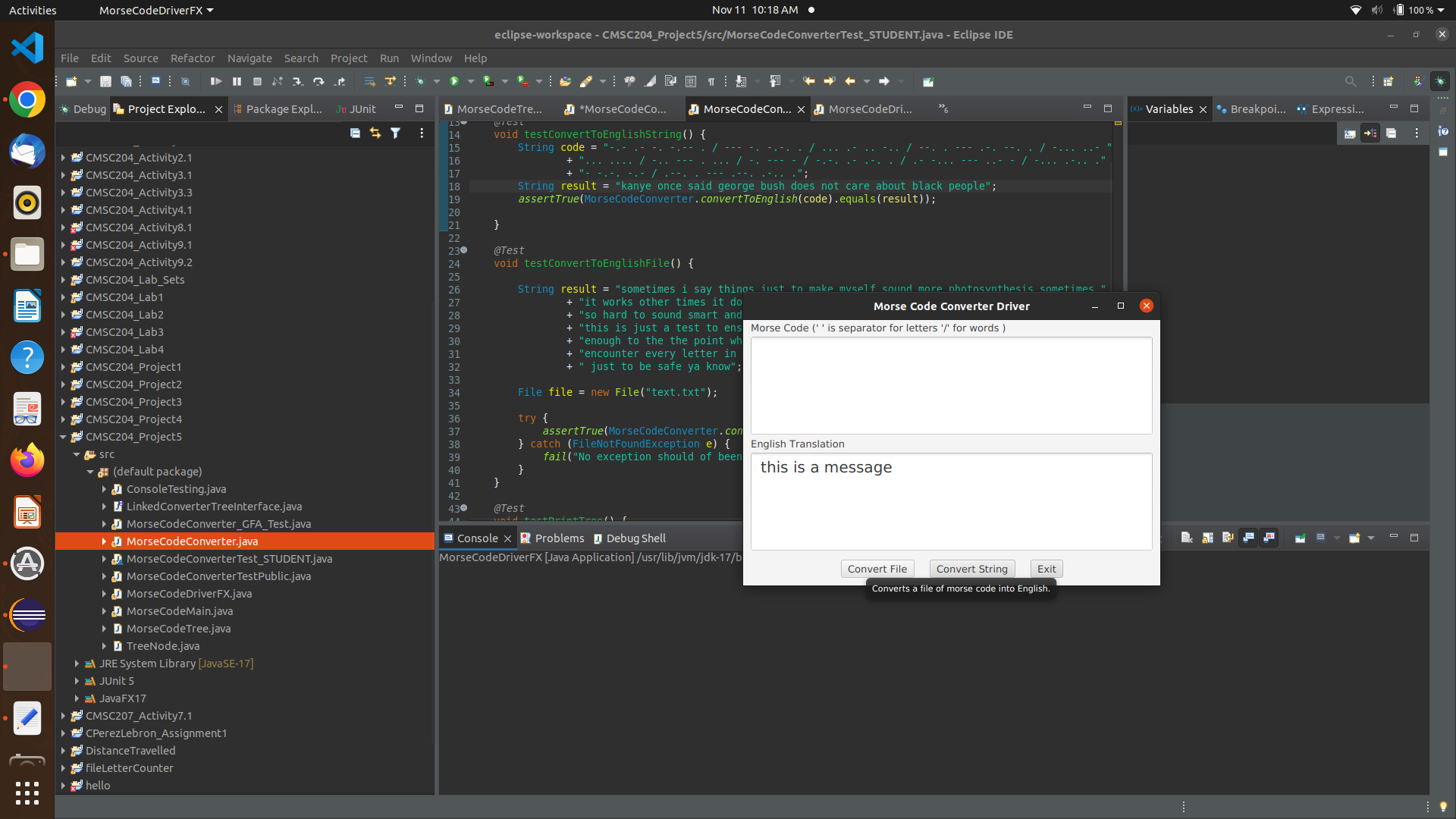
# Test Cases

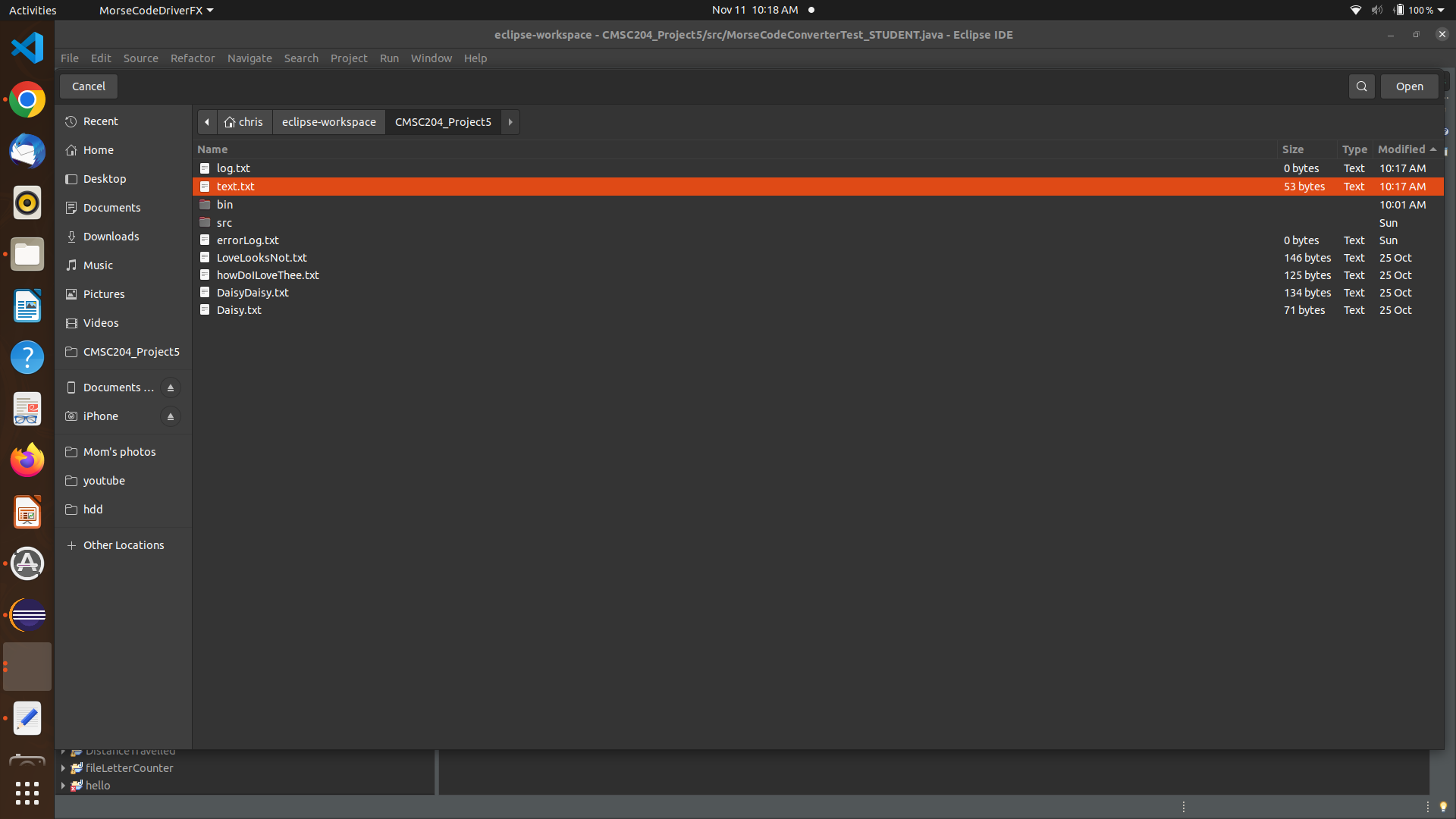
case: - .... .. ... / .. ... / .- / -- . ... ... .- --. . [a normal message no unexpected or unknown characters]

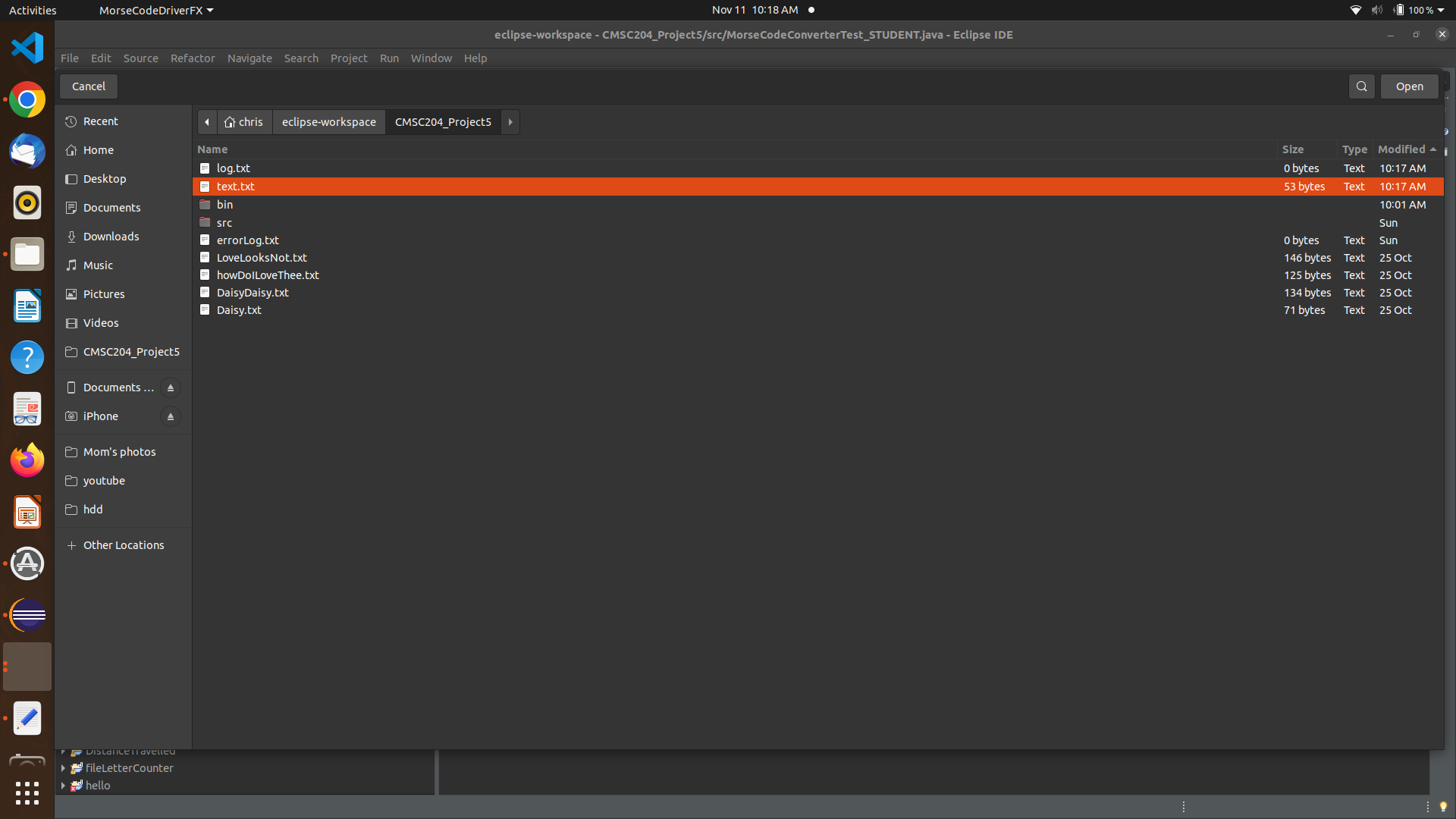
expected: this is a message

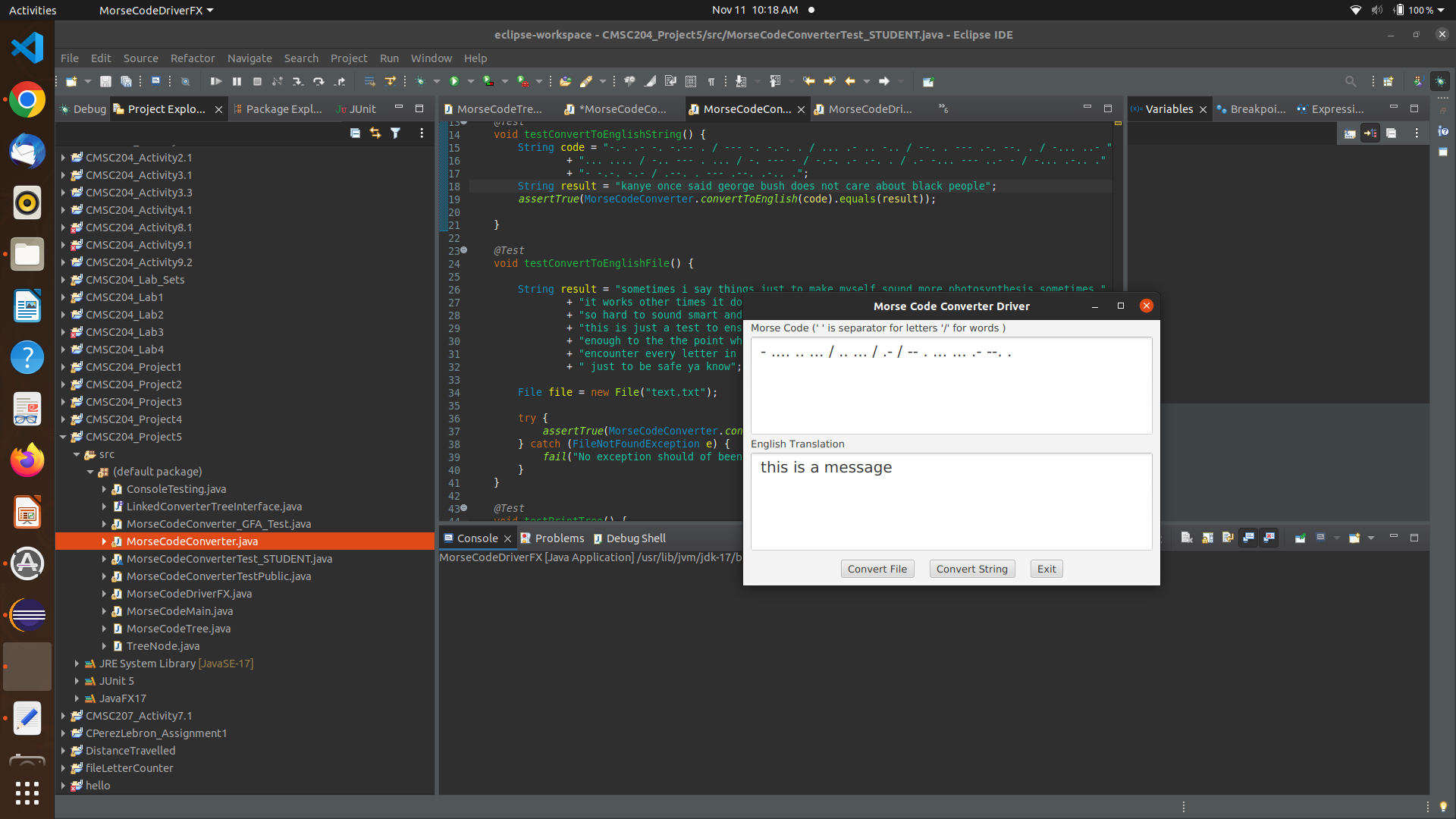
result: this is a message [as expected]







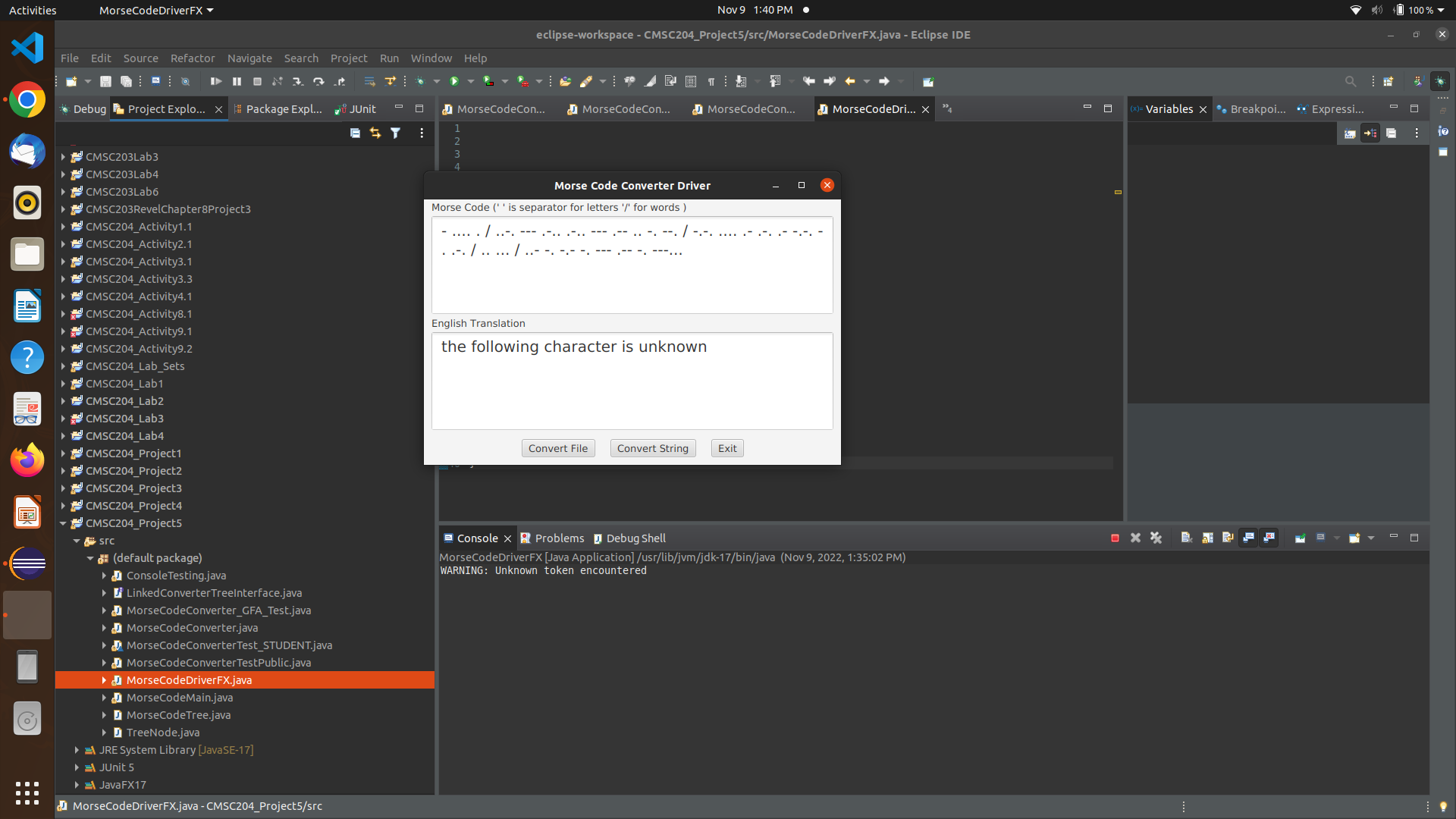


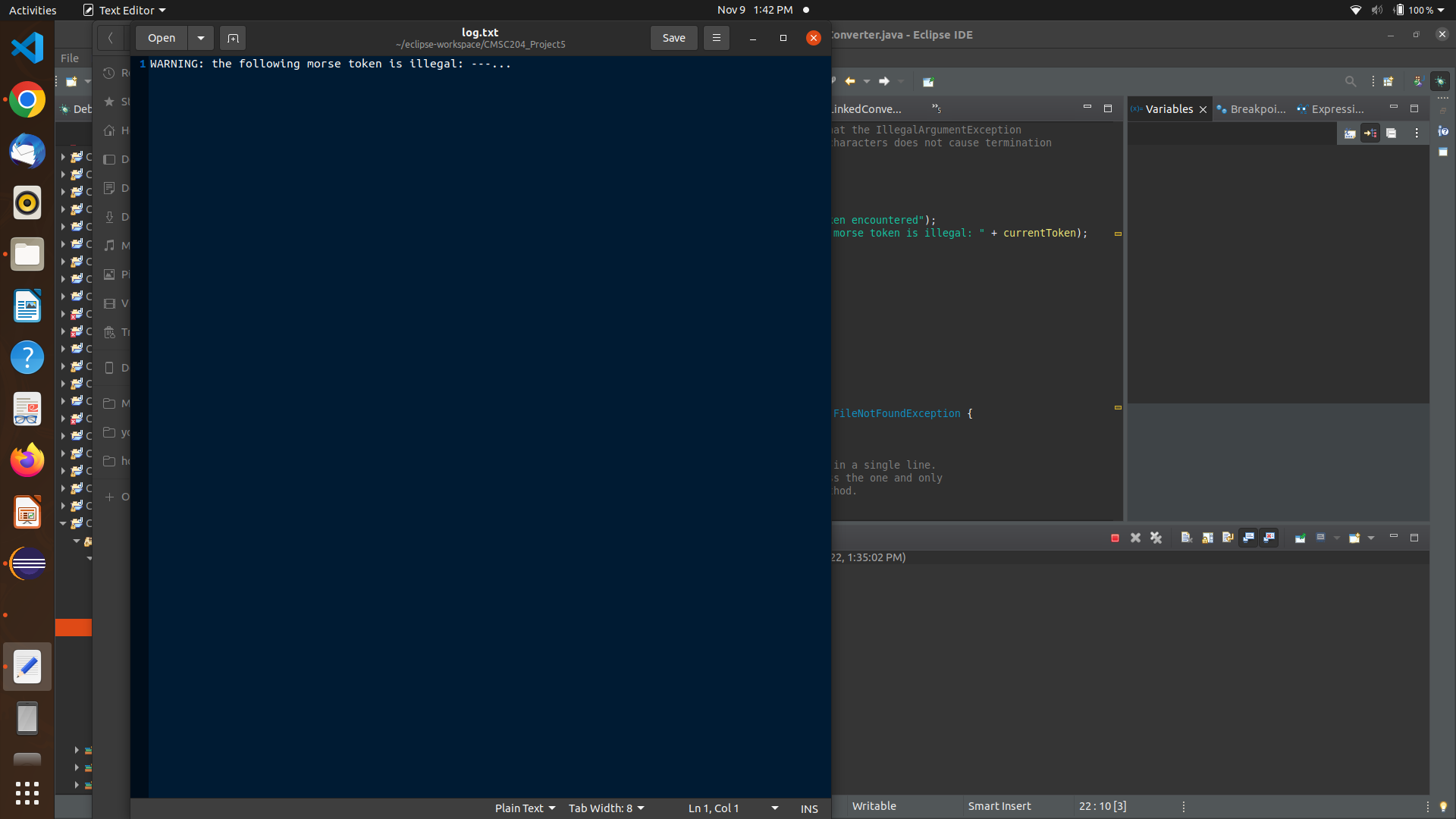


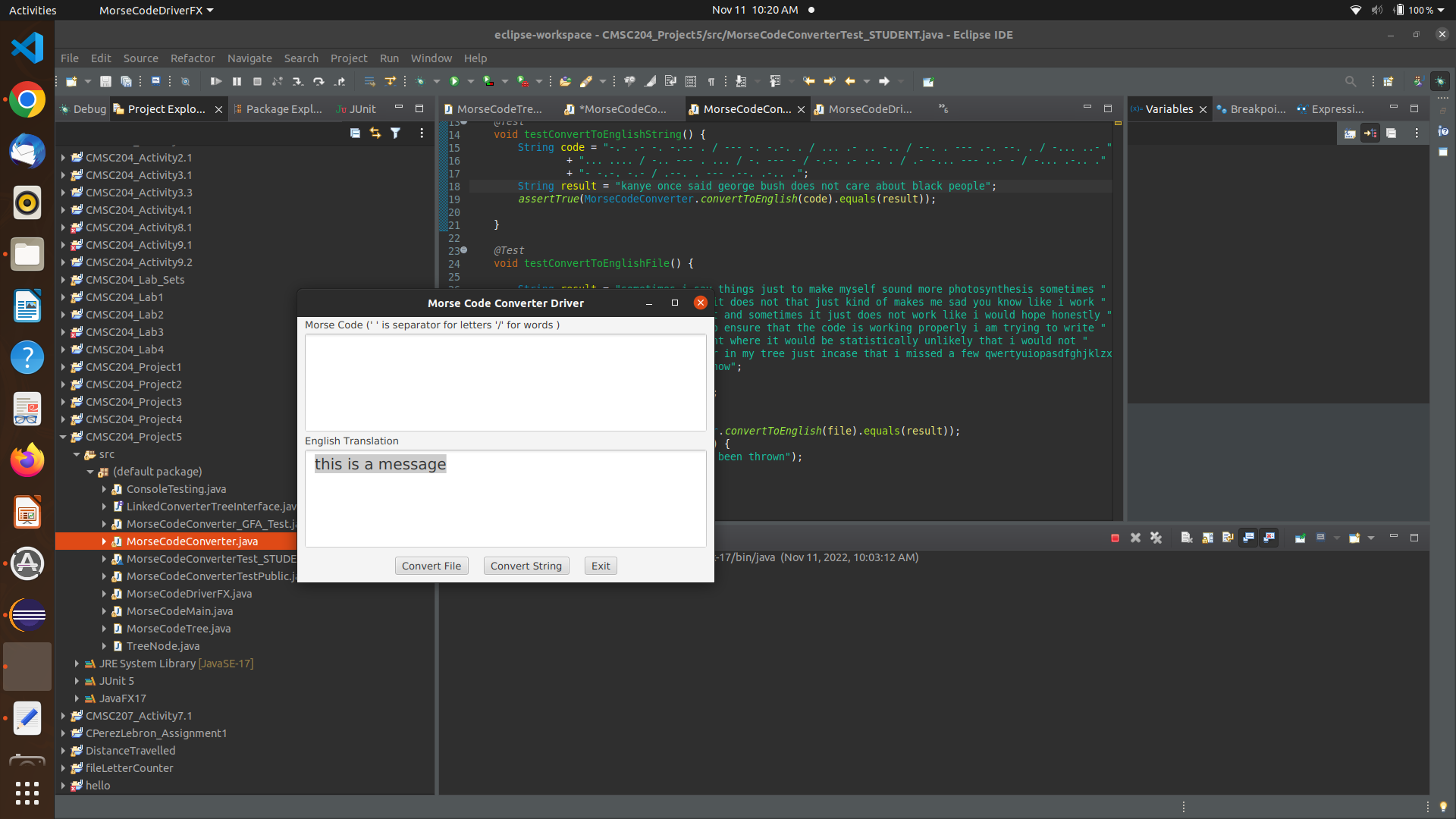
case: - .... . / ..-. --- .-.. .-.. --- .-- .. -. --. / -.-. .... .- .-. .- -.-. - . .-. / .. ... / ..- -. -.- -. --- .-- -. ---... [“the following character is unknown:” we do not support the last character, the colon.

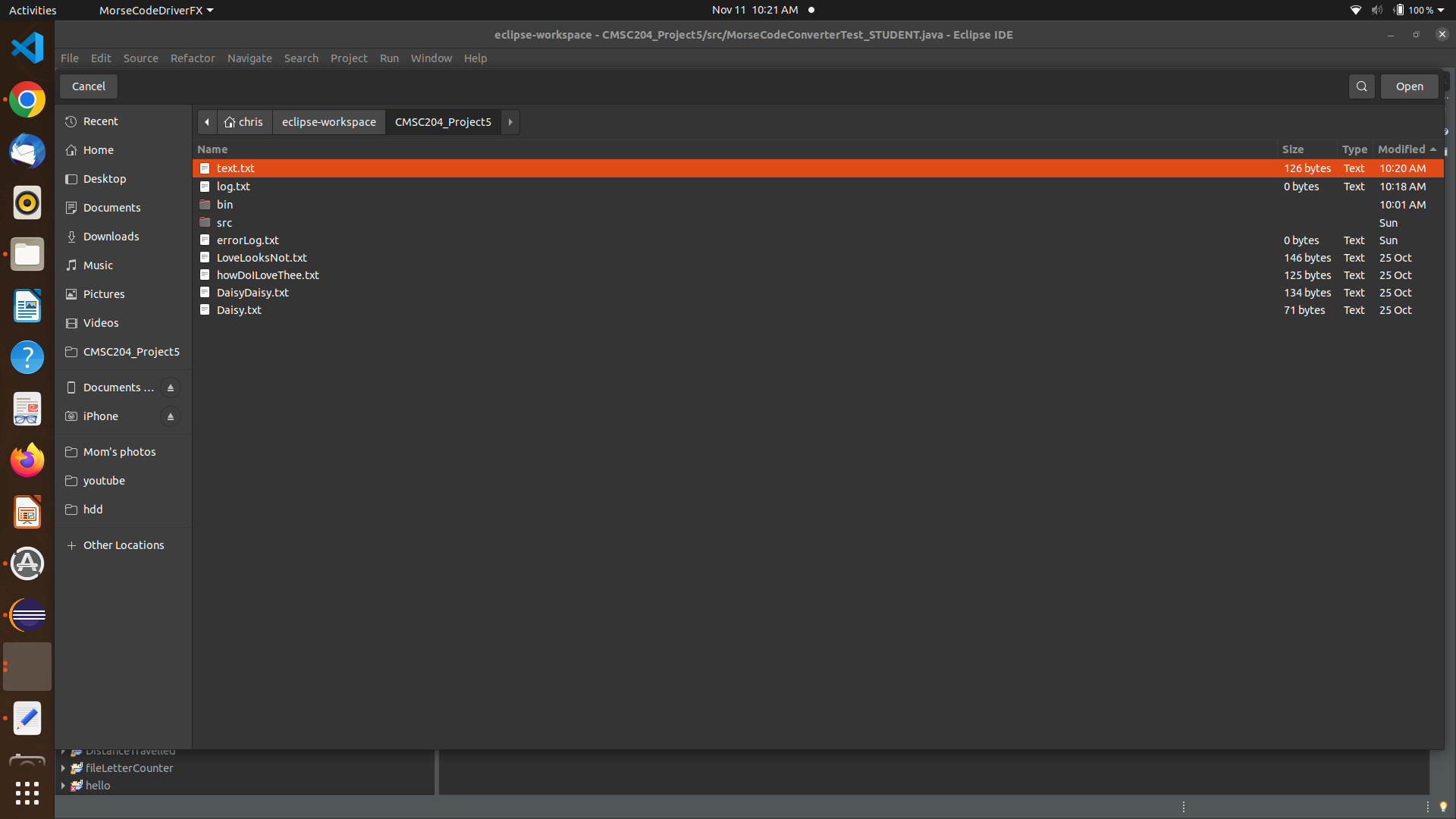
expected: I expect my error handling to kick in but I am not sure how the GUI will deal with it

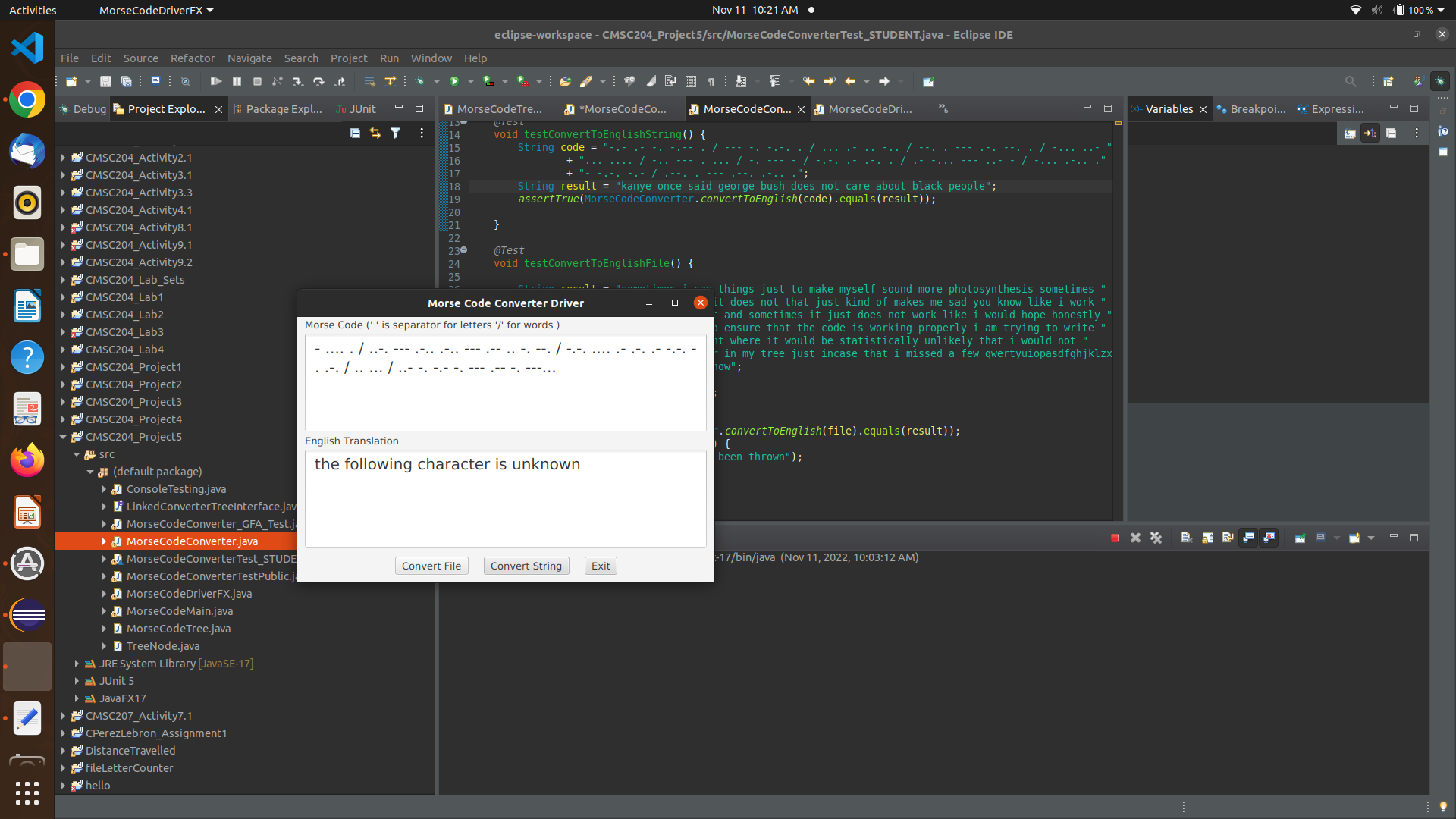
result: GUI did not break, and my error handling kicked in. I warning was sent to console along with a more detailed message in the “log.txt” file in the project’s root directory.

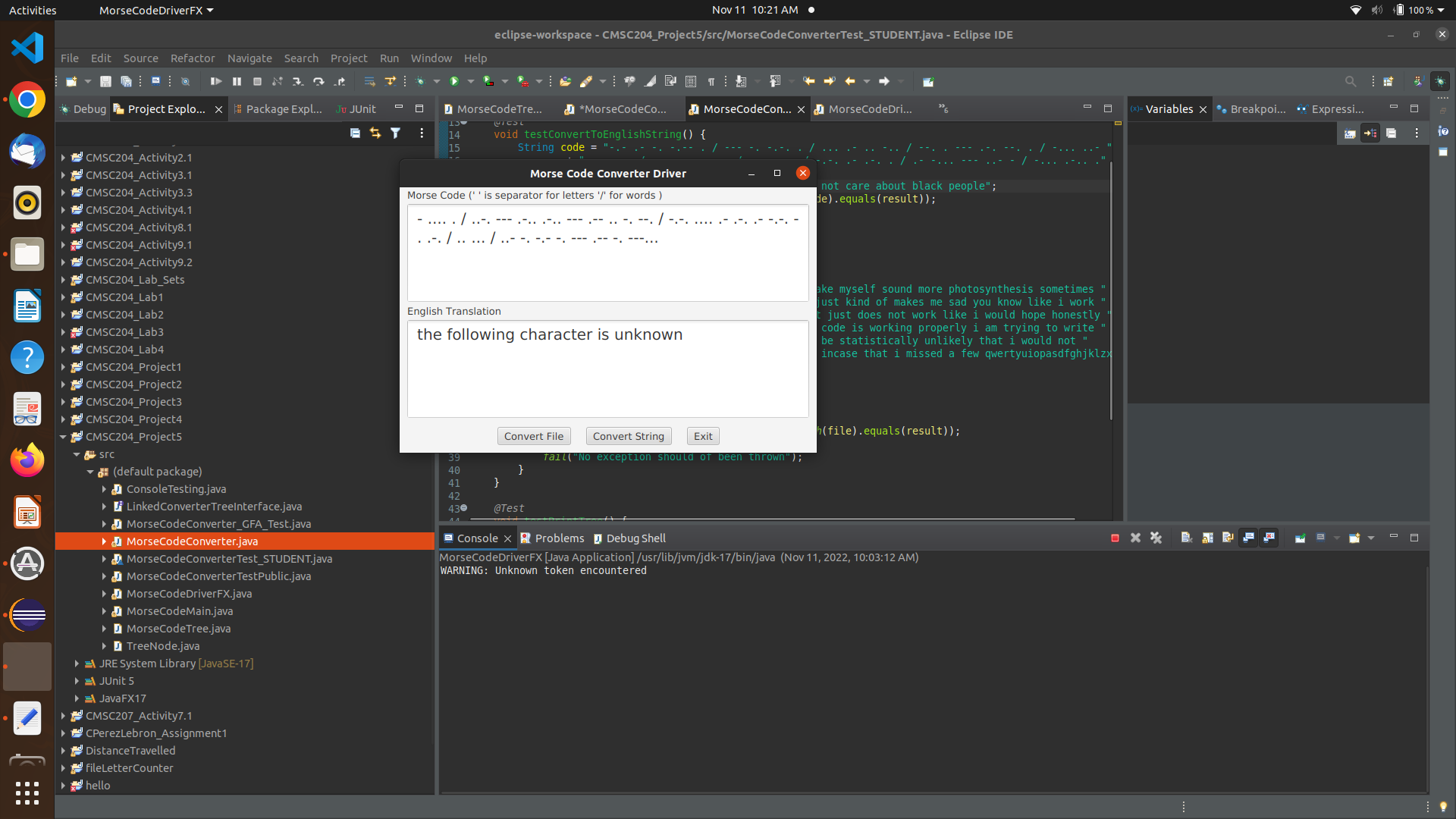


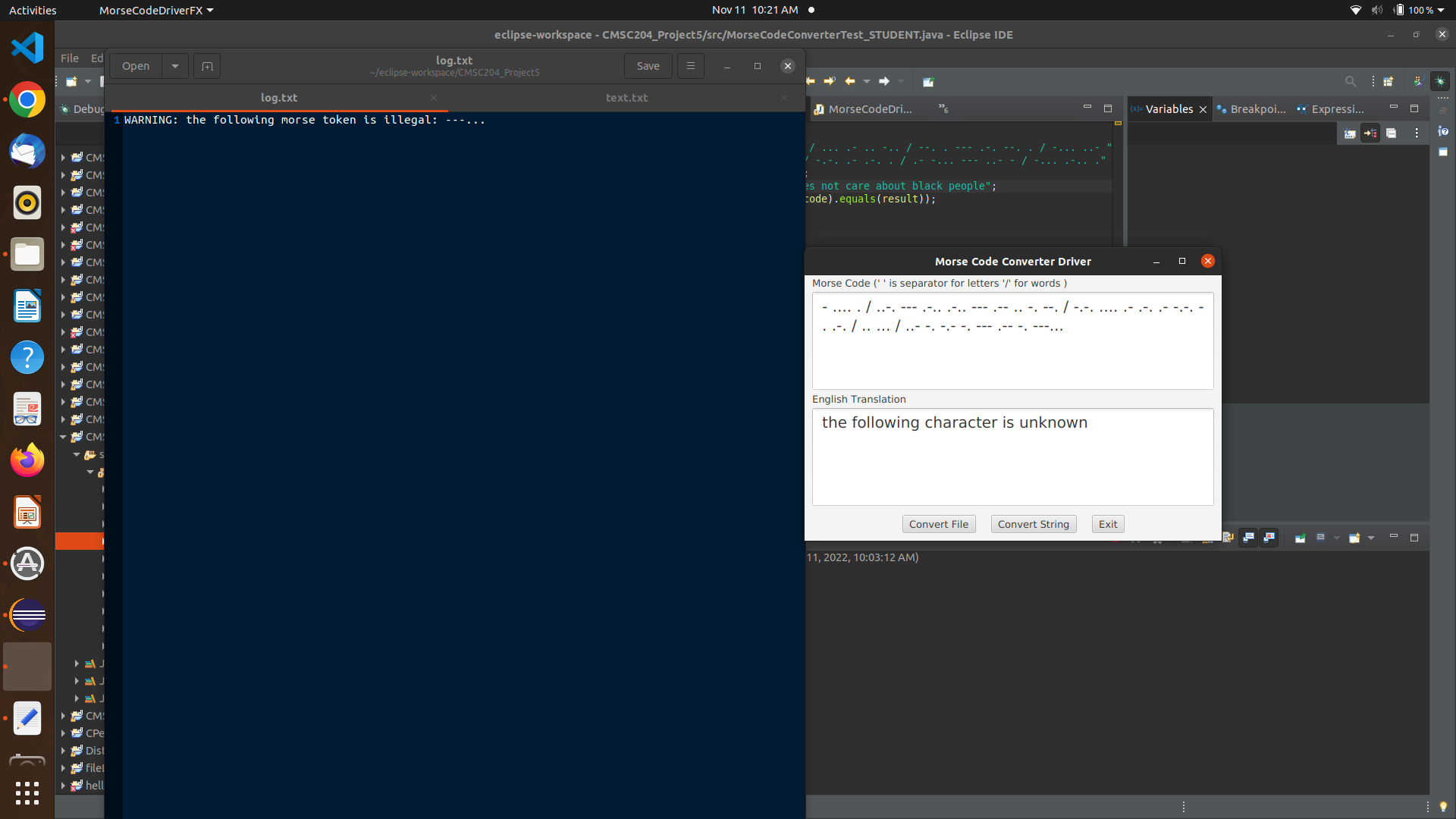








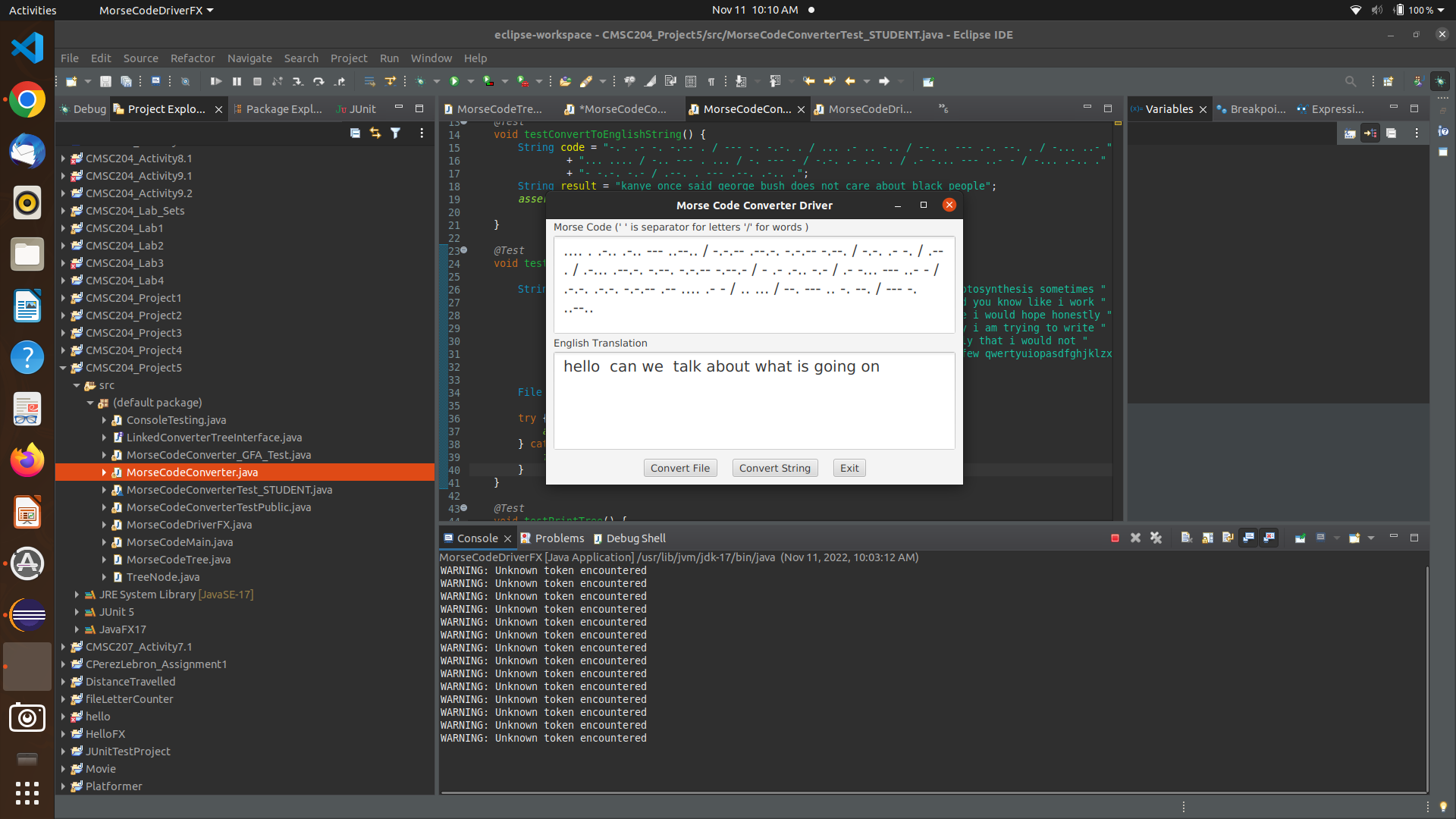


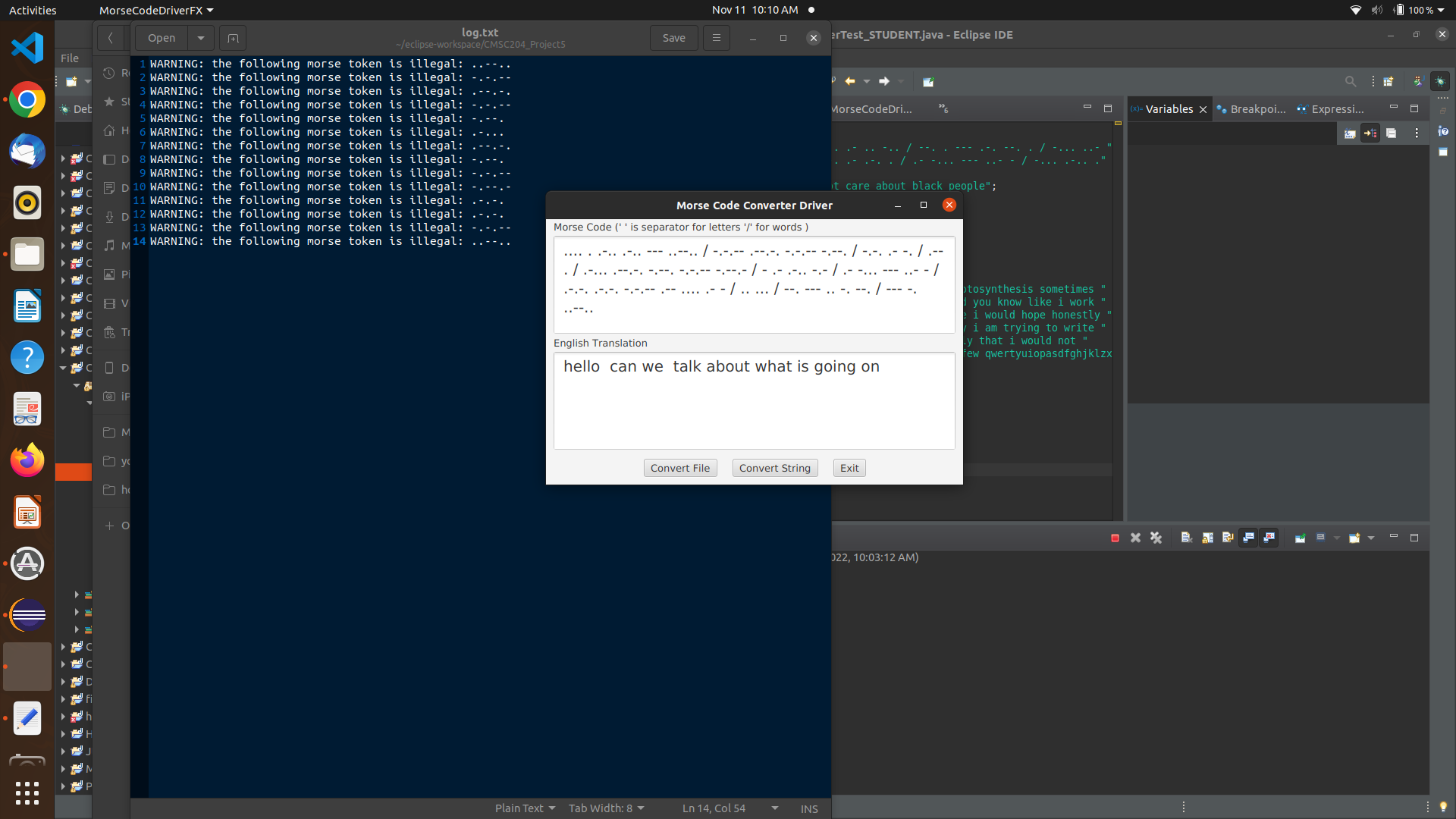


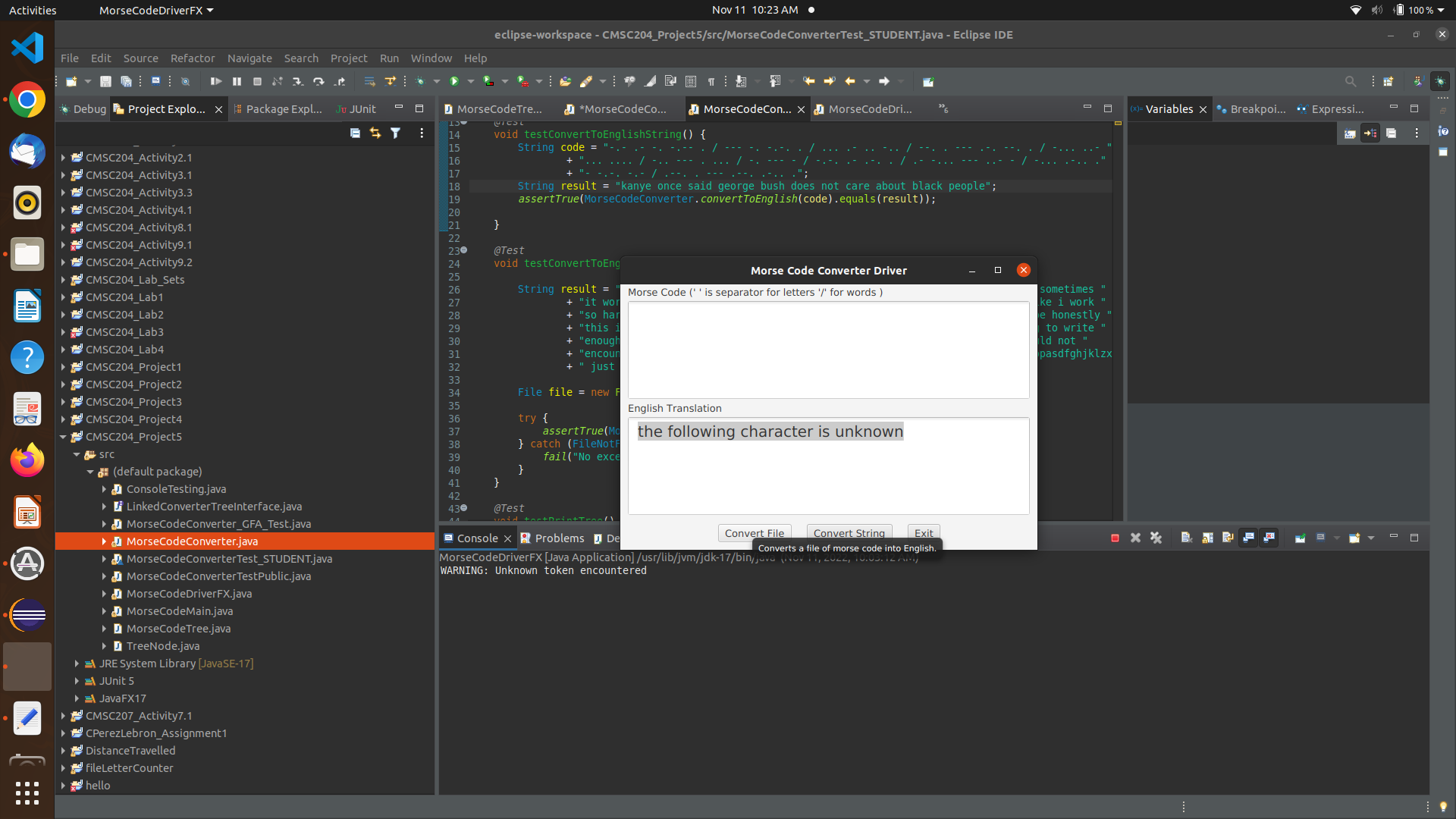
Case [a message with multiple unknown characters throughout the string] .... . .-.. .-.. --- ..--.. / -.-.-- .--.-. -.-.-- -.--. / -.-. .- -. / .-- . / .-... .--.-. -.--. -.-.-- -.--.- / - .- .-.. -.- / .- -... --- ..- - / .-.-. .-.-. -.-.-- .-- .... .- - / .. ... / --. --- .. -. --. / --- -. ..--.. [hello? !@!( can we &@(!) talk about ++!what is going on?]

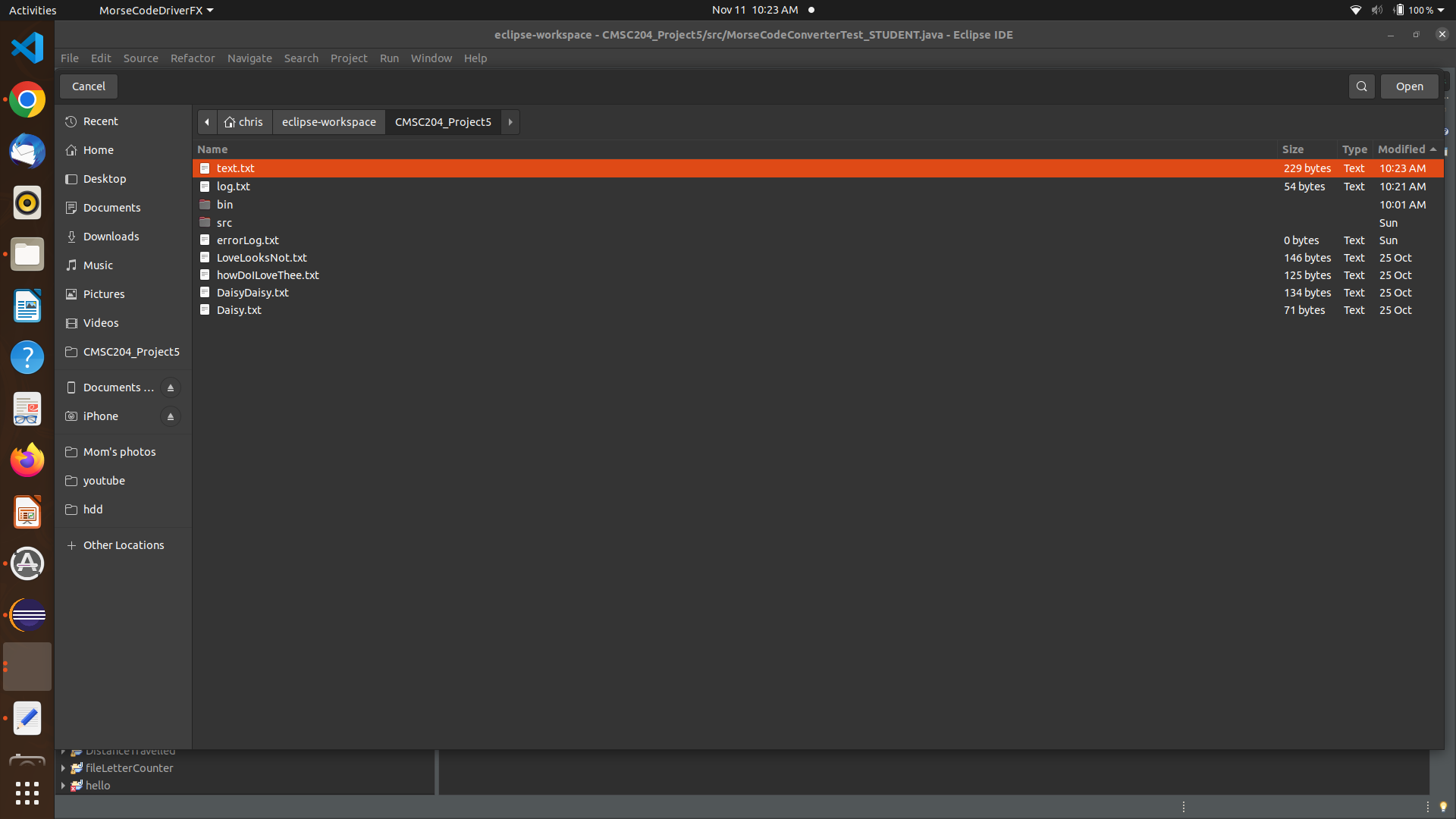
expected: error handling to kick in and any unknown characters to be skipped

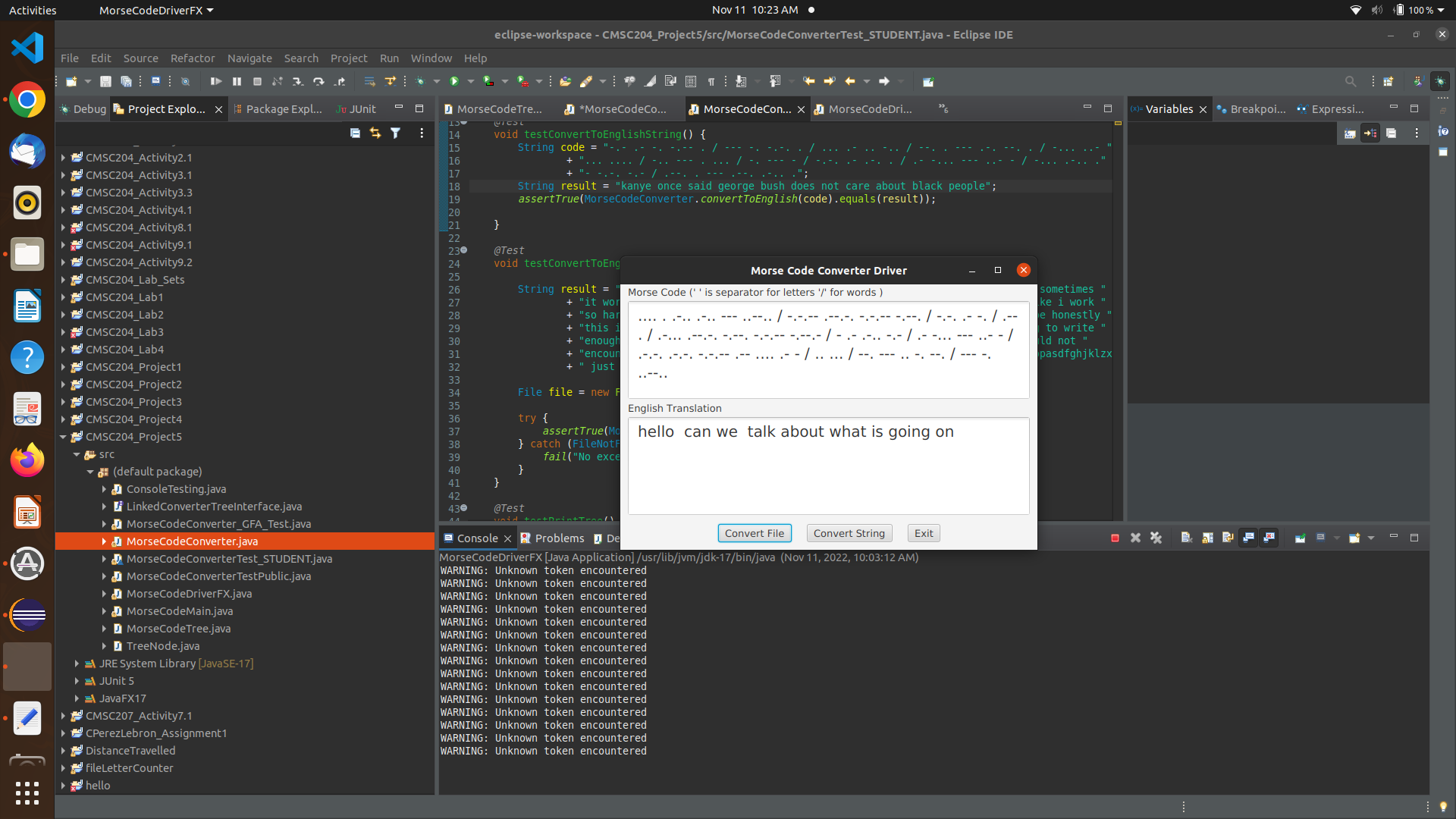
result: as expected

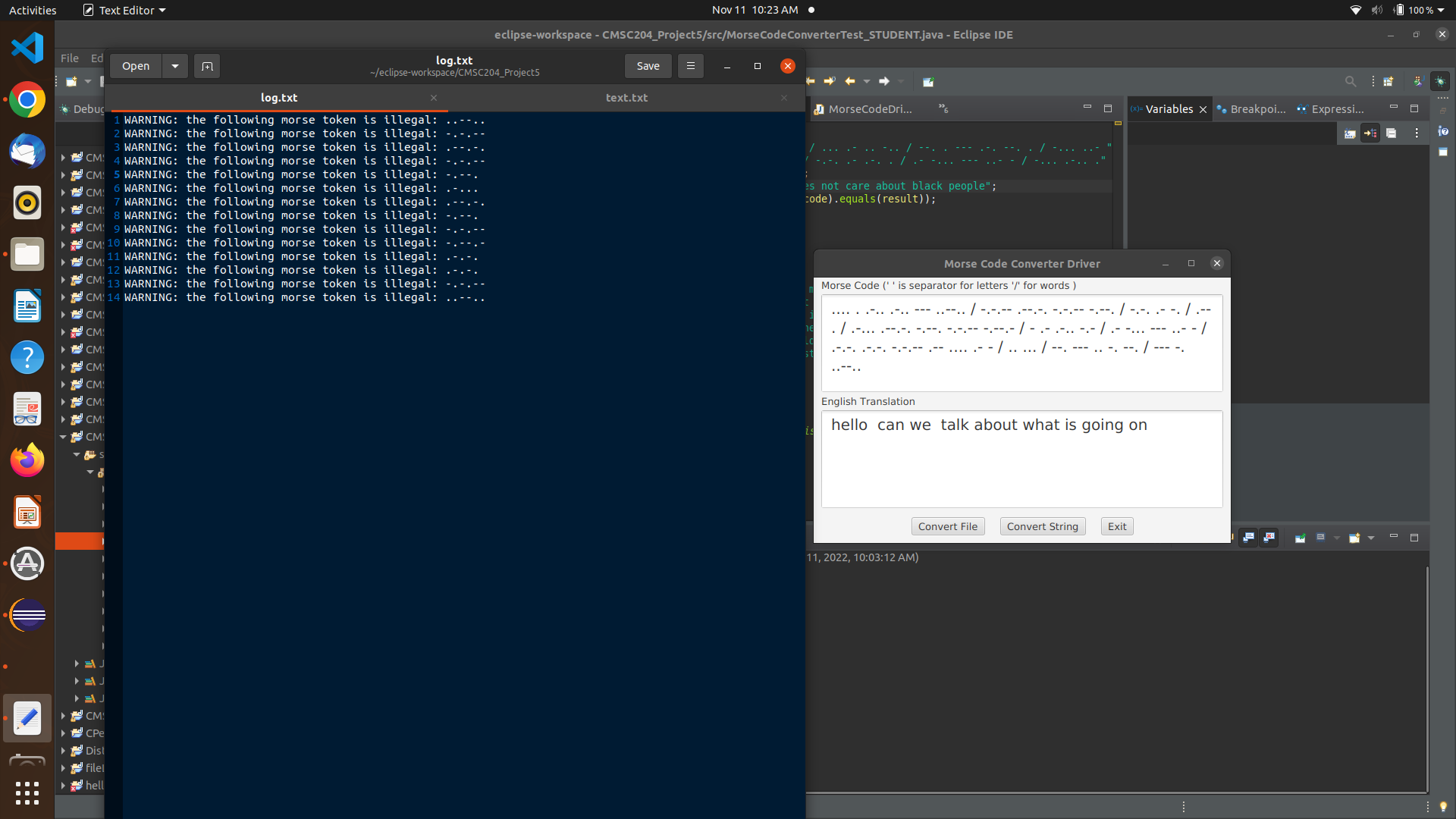










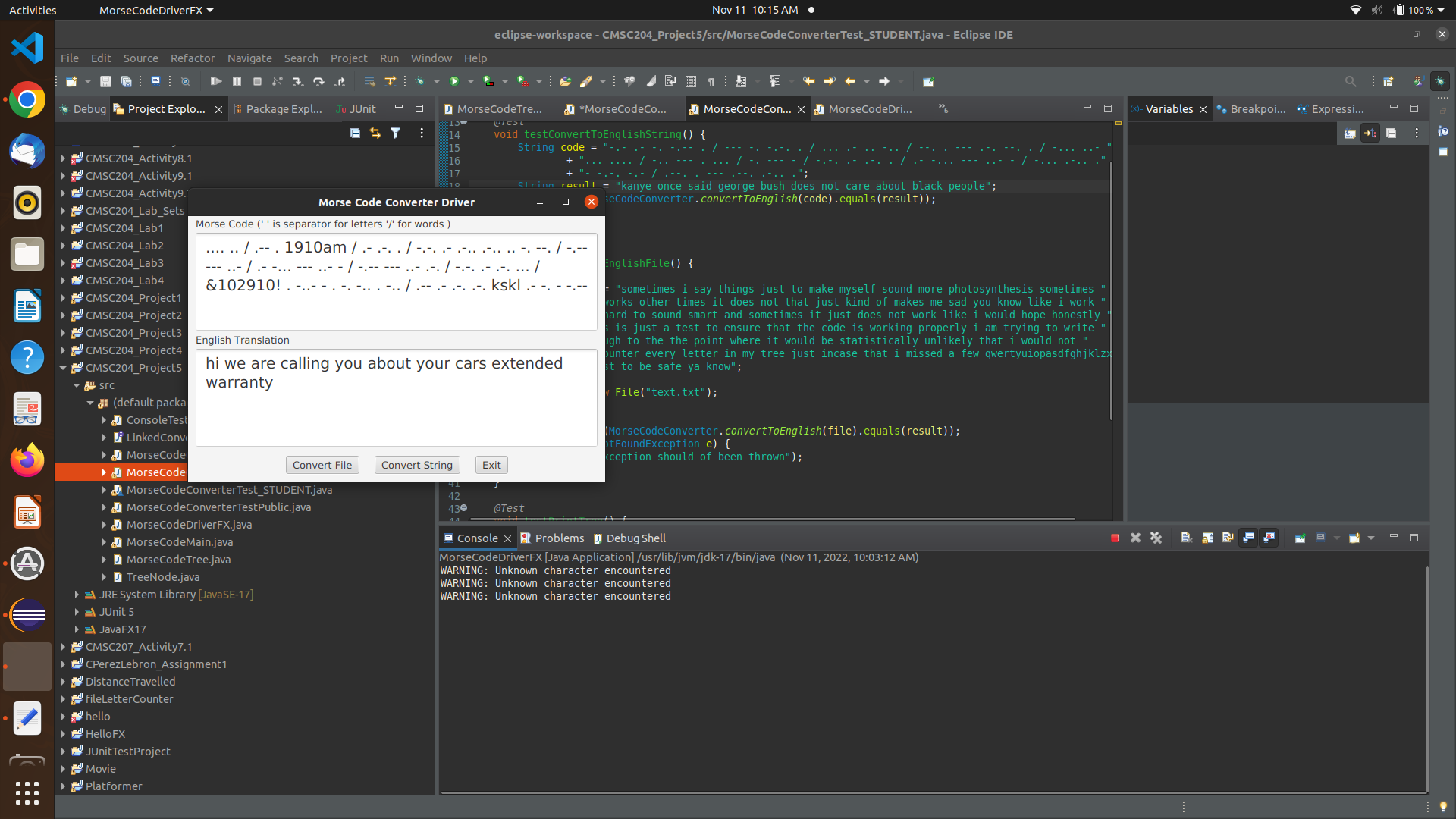


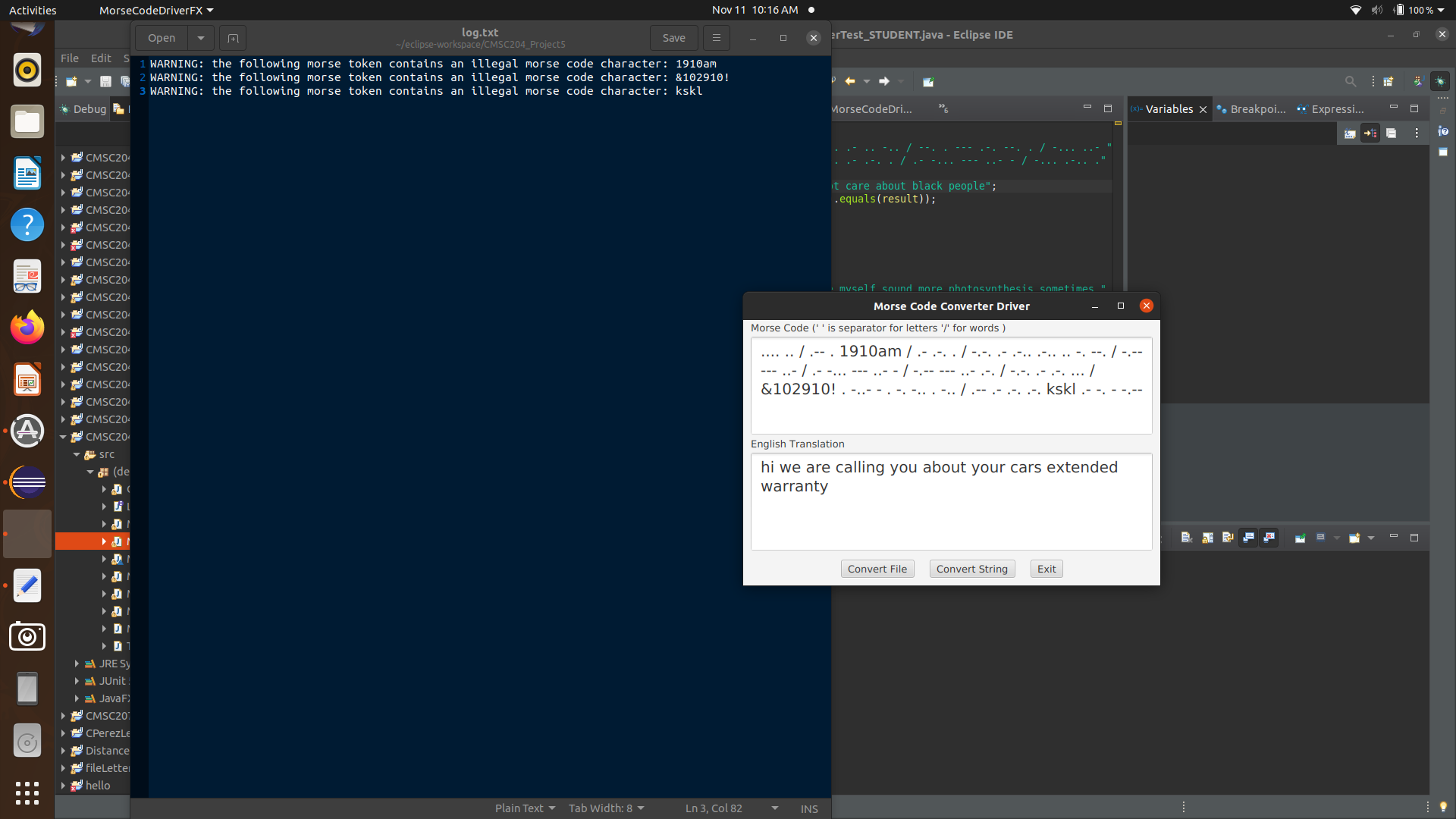
case: .... .. / .-- . 1910am / .- .-. . / -.-. .- .-.. .-.. .. -. --. / -.-- --- ..- / .- -... --- ..- - / -.-- --- ..- .-. / -.-. .- .-. ... / &102910! . -..- - . -. -.. . -.. / .-- .- .-. .-. kskl .- -. - -.--

[hi we are calling you about your car's extended warranty]

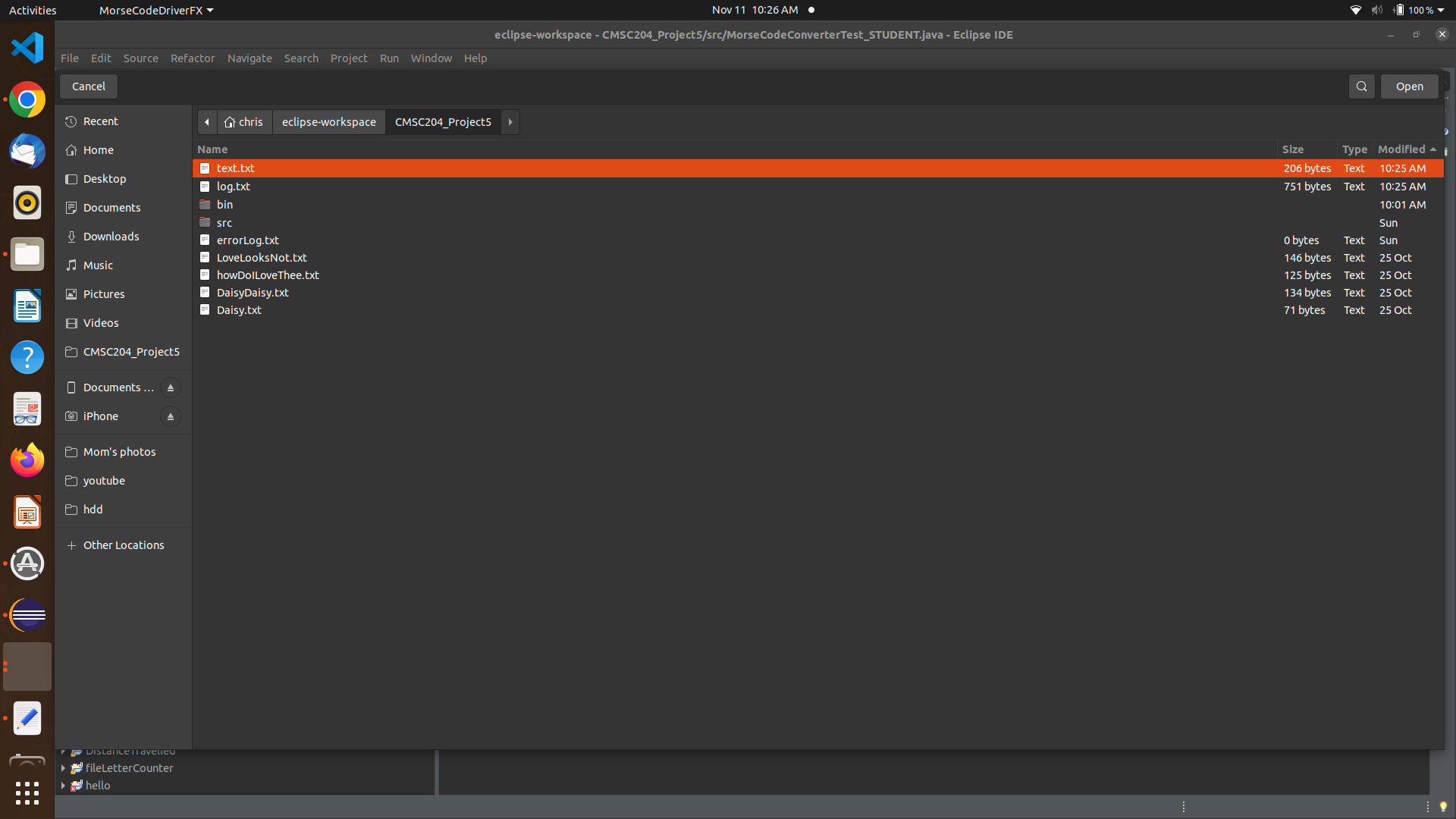
expected: I specifically placed unknown characters BETWEEN morse code tokens so that no character from the original morse code string would be skipped [with my error handling implementation: if an unknown morse code char is encountered within a morse code token, the entire token is skipped.]. Hence, my expectation is to see “hi we are calling you about your car's extended warranty”

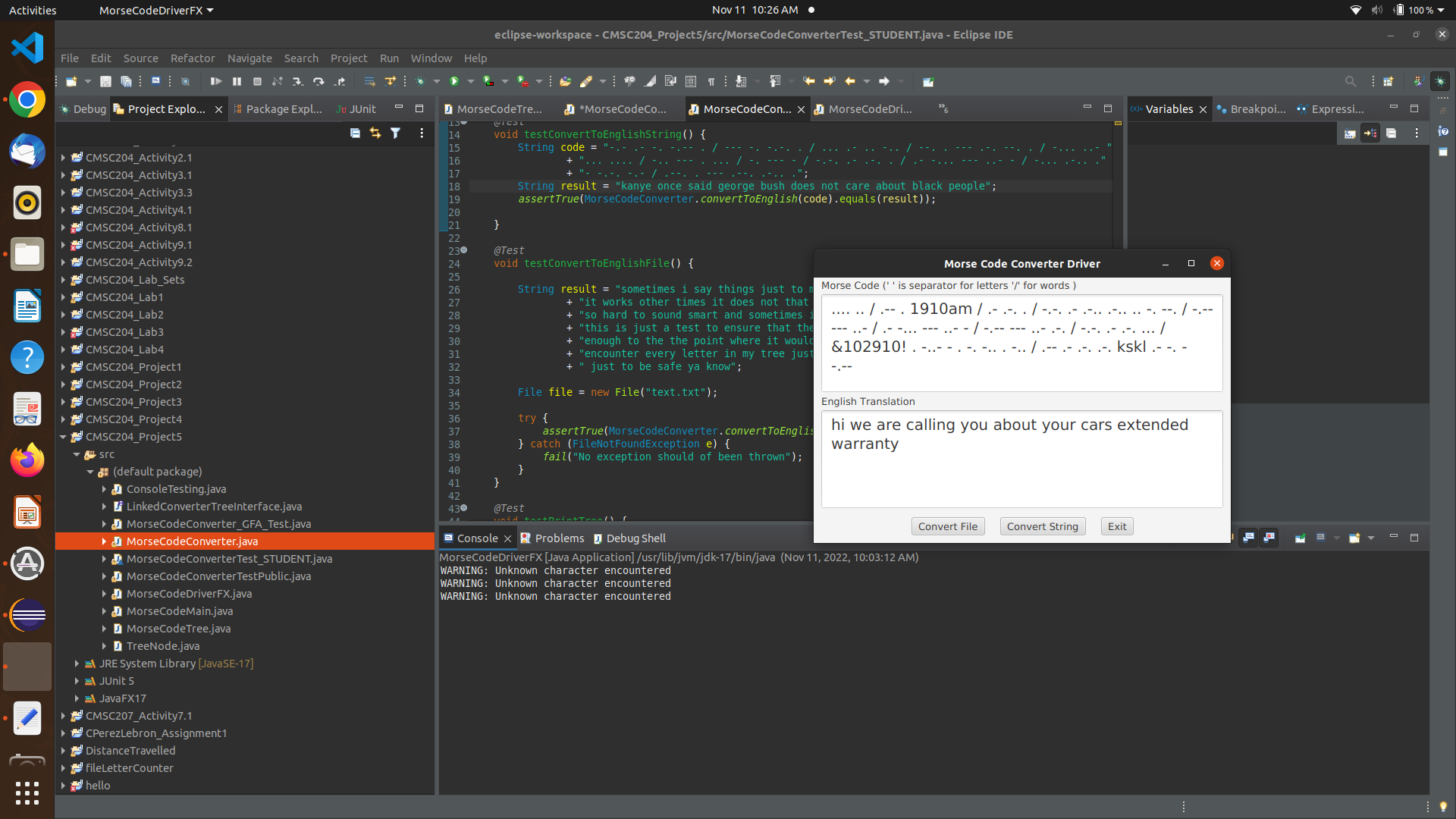
result: as expected

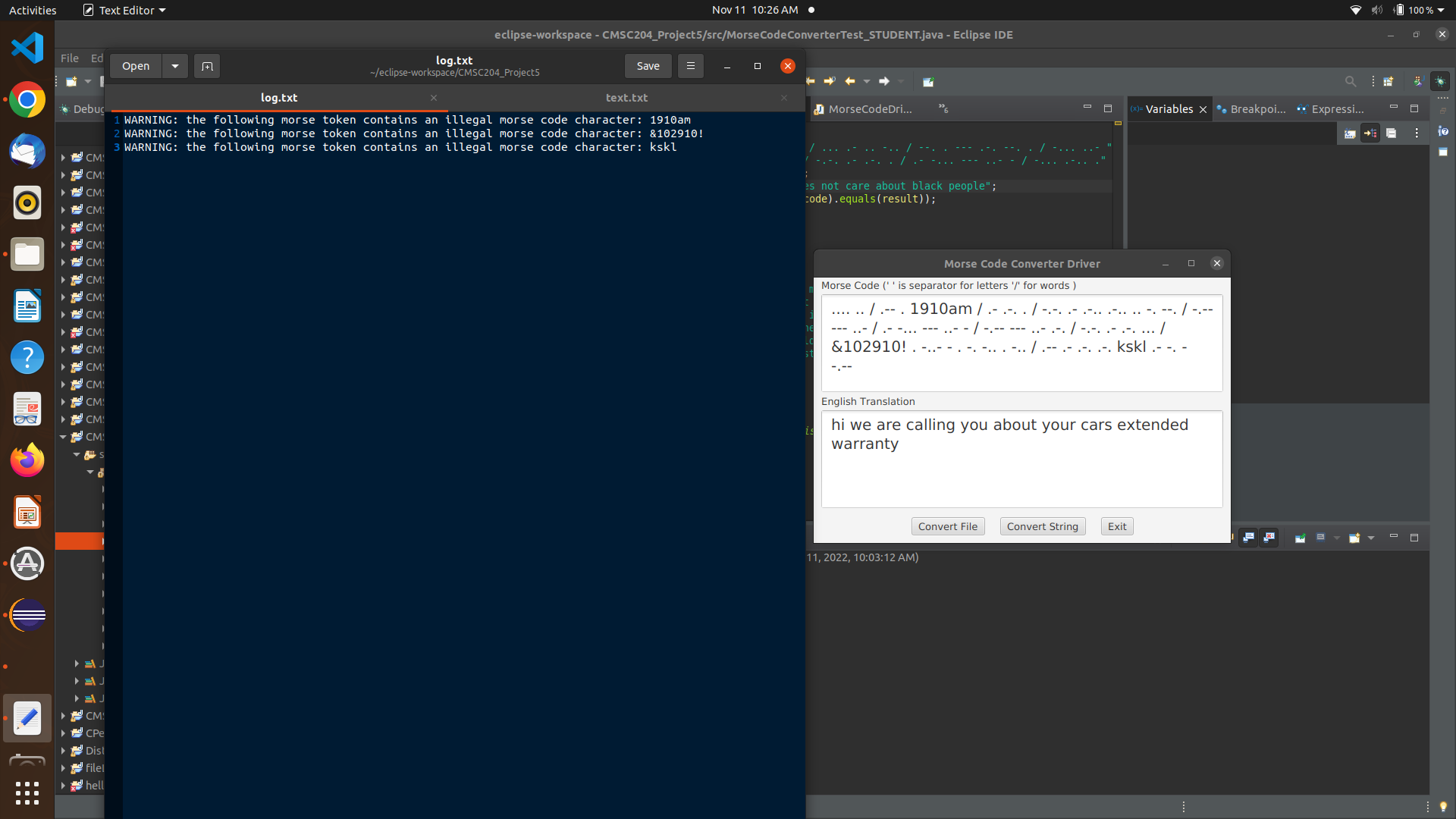




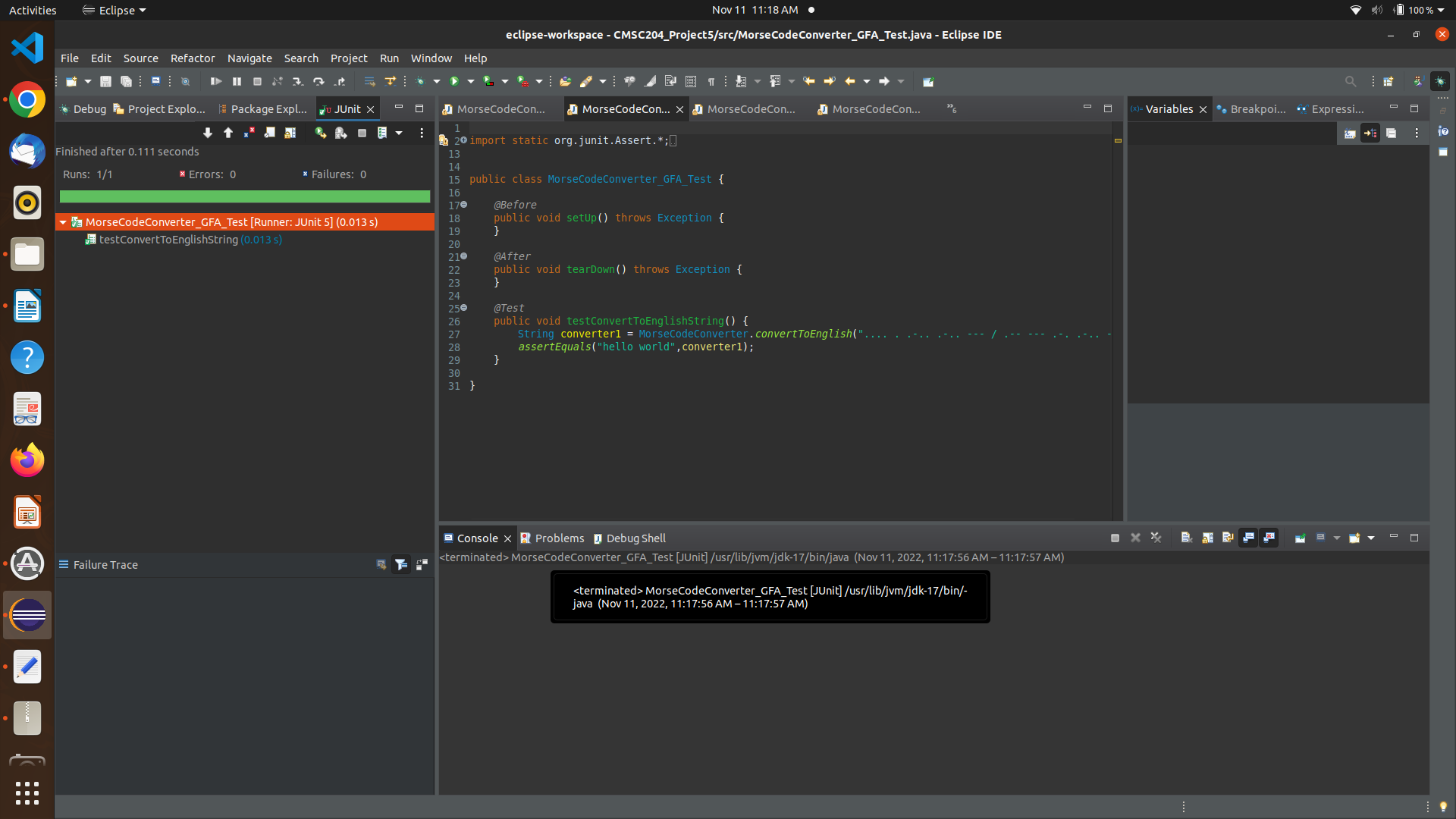
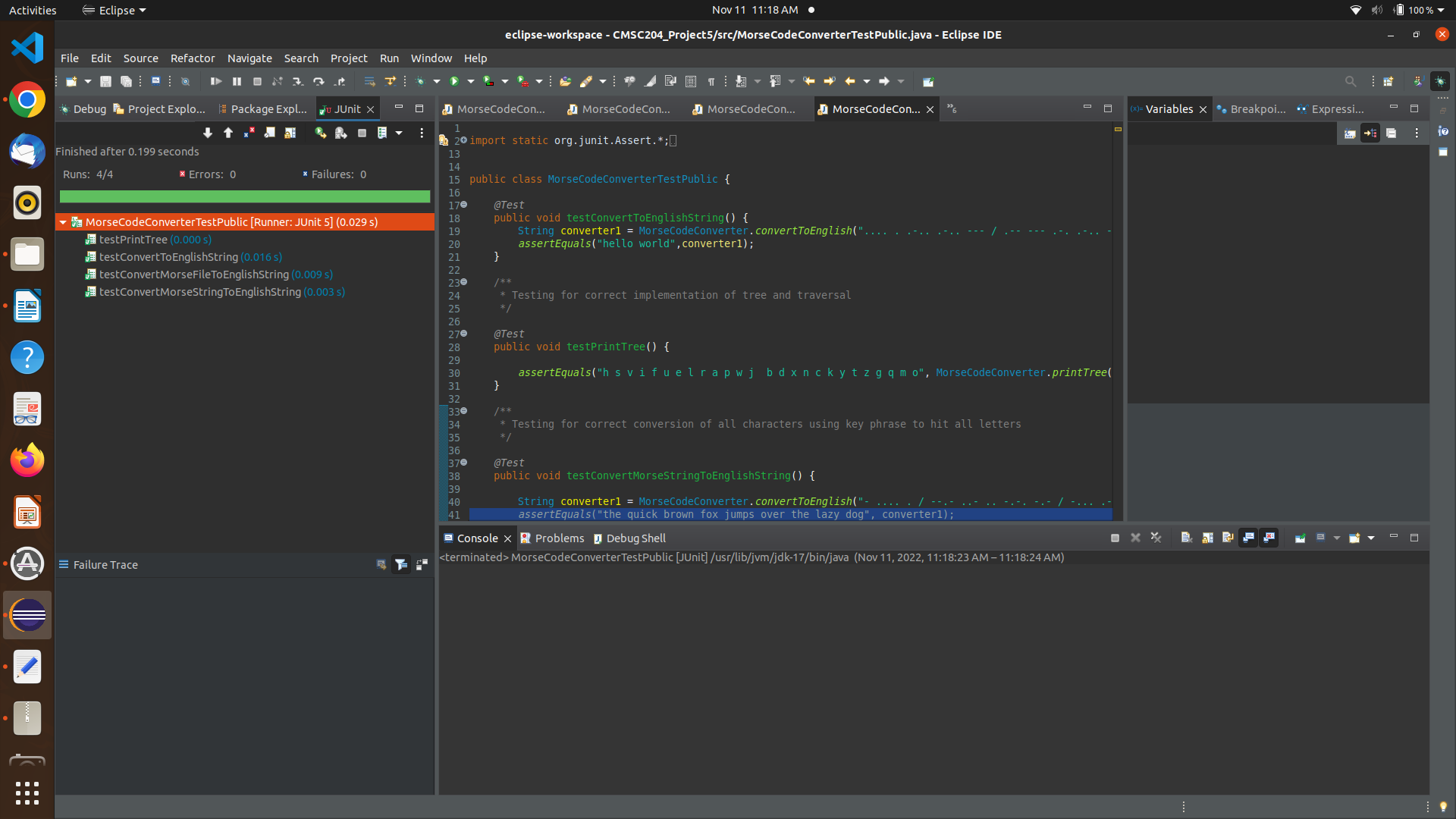








# JUnit Test Screenshots



# Github Screenshot