# Approach, Design & Algorithm

To be quite honest, it was a bit challenging to come up with a high-level plan for this project because of the vagueness of the “Write-up” document provided and the lack of any java docs. Nonetheless, I analyzed the provided JUnit tests, and GUI to get a better idea of what the project wanted me to do. I would say the hardest part of this project was deciphering what the specifications were asking for. Otherwise, the project went pretty smoothly.

The implementation for the CourseDBElement class was pretty straightforward. I implemented a constructor that allows the client to insert values for all the fields and defined get methods for all the fields and set methods for all the fields but the crn. I chose to override the toString method provided by the Object class to make converting CourseDBElement objects into strings easier in the future. This toString method would come in handy for the showAll method in particular. There was virtually no info on how the compareTo method will be used within our implementation so I wasn’t entirely sure what to do with it. I left that implementation for later to see if coding other parts of the projects would provide some clarification. Ultimately, I got through the entire project without ever finding any need for this method or the comparable interface. As alluded to in the notes of the compareTo method I implemented, I think the closest use for the compareTo method would be as a method to compare the CRN of two course objects. However, there is no use, within the scope of this project, for knowing whether a course object is less than or greater than another. Since this project primarily deals with comparing CRN values I decided to implement the compareTo method in a way that compares the CRN values between the two objects. However, I do not use this method as again, the only valid use for it in this project is as a method to check if two CRN’s are equal, but using a method named “compareTo” to only check equality would make my code confusing to read. Moreover, I later realized that FXMainPane of the GUI component wanted a default constructor for this class. I personally see no use in such a constructor and think it increases the chances of uninitialized values but I implemented it as desired by the FXMainPane class. Furthermore, as I was reviewing the FXMainPane class I saw that it makes use of a setCRN method. I think it is a bad idea to “set” the crn field to a new value but the FXMainPane required it to run properly, thus I implemented it.

My plan for the CourseDBStructure class was pretty simple. On a basic level, the objective was to implement a hash table using LinkedLists instead of nodes to represent the buckets. I just needed to create a constructor, an add method, a get method, a showAll method, and a getTableSize method. The initial code was pretty straightforward to implement and went smoothly. The challenges arose trying to adhere to the JUnit test’s desires. Many of which I think are bad ideas.

Similarly for the CourseDBManager class, the initial plan was extremely simple and straightforward. However, I had an issue passing the JUnit test’s test case for the showAll method. This method clearly enforced a strict ordering of the elements but It was hard to find a way to make the iteration order of my hash table look like the iteration order the JUnit case was testing for.

# Additional Features

For my CourseDBManager class’s readFile method I implemented code to an error log file. This implementation was fairly straightforward, whenever an exception arises trying to read a line of course data a message indicating the line number that caused the error is printed to the “errorLog.txt” file. Once the readFile method is complete the errorLog.txt file will contain a list of the lines that were not processed properly and caused some sort of exception.

I created a class that would generate a text file filled with dummy course data to aid in testing. This class, named CourseDataGenerator, is really simple. First I had to find a list of first names from the internet, once I found that data I placed it in a file named “FirstNames.txt” for later use. I did the same for a text file named “LastNames.txt”. Once I had source data for last names and first names it was just a matter of generating random letters and numbers in the correct locations using the expected format. Despite its simplicity, having this dummy data is pretty useful for testing, as you will see in the test case section.

# Learning Experience

**What did you learn?**

Honestly, I do not think I learned anything new from this project. However, it is decent practice. I say decent because I spent a lot more time deciphering the project expectations and working around what in my opinion were bad expectations than actually programming.

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

The only issues I encountered were in relation to project specifications and expectations. Other than that, my project went smoothly.

**What would you have done differently?**

I wouldn’t do anything differently. I would, however, suggest either updating this project assignment or swapping it with a different project, not because the project’s contents are hard but because in my opinion the project is poorly written.

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

Databases are everywhere, but I doubt I will be implementing my own database in the manner we did in this course. I would suspect I would use some pre-existing database for most projects. On the other hand, understanding hashing and its use in a dictionary are likely far more important than the overall goal of making a database.

**Questions**

I have no questions regarding the content of this project.

**Assumptions**

I definitely made assumptions because of the lack of detail in the project specifications but none of them are far enough from reasonable to be note worthy. I did, however, discuss the assumptions I made with the compareTo method above in the design portion.

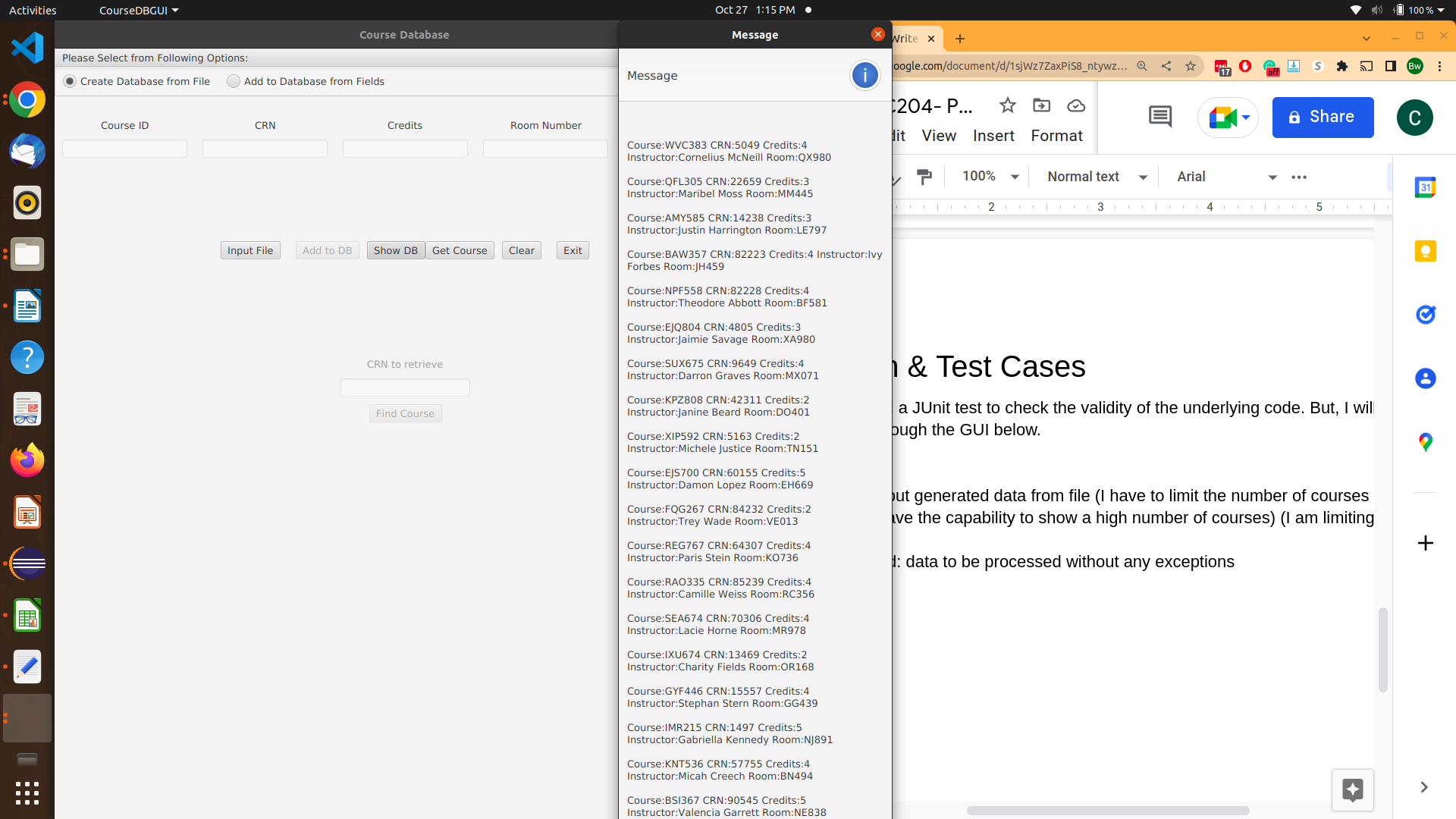
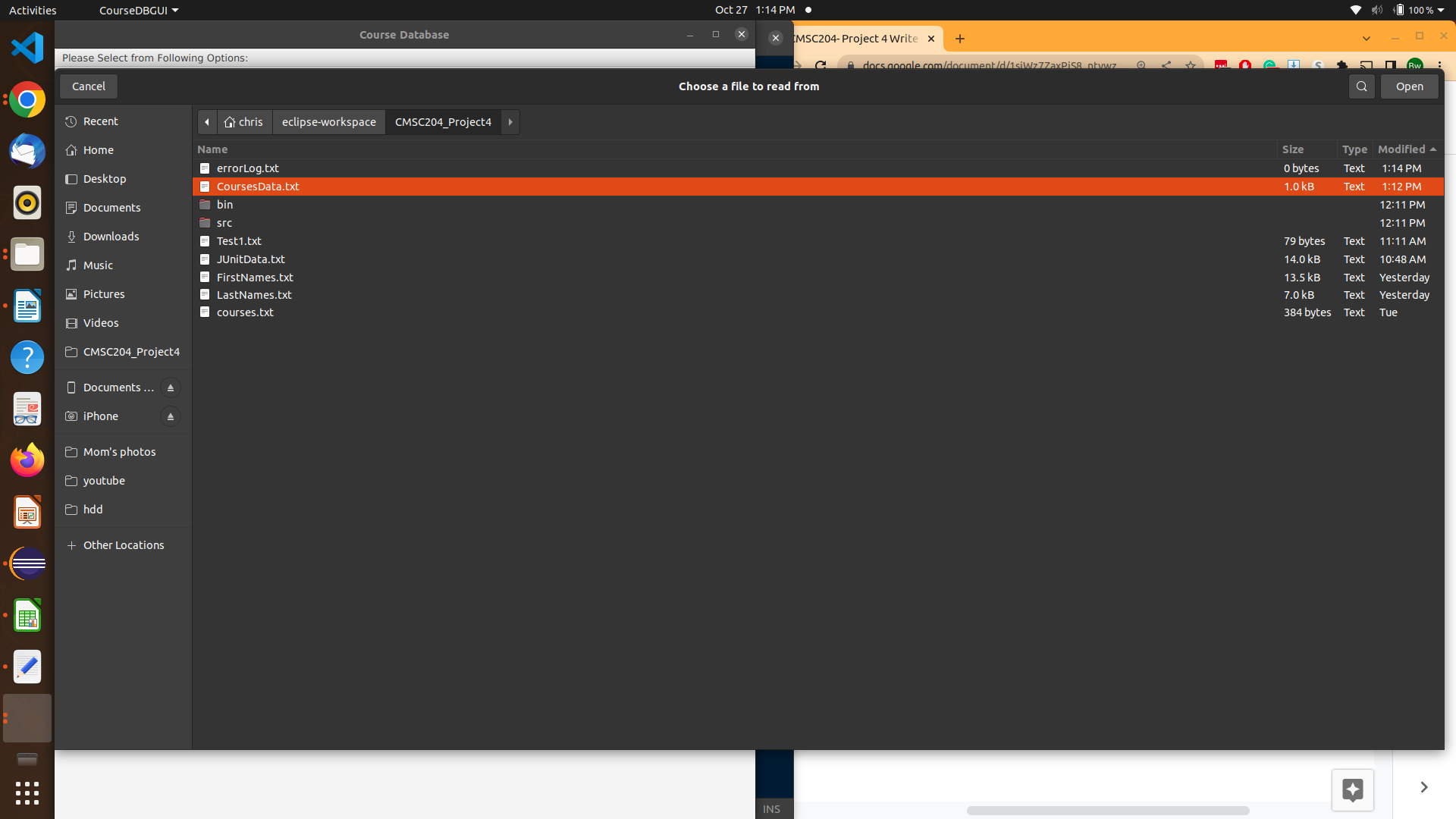
# Test Plan & Test Cases

I created a JUnit test to check the validity of the underlying code. But, I will also provide more testing through the GUI below.

case: input generated data from file (I have to limit the number of courses because the GUI does not have the capability to show a high number of courses) (I am limiting it to 30 courses)

expected: data to be processed without any exceptions

result: as expected

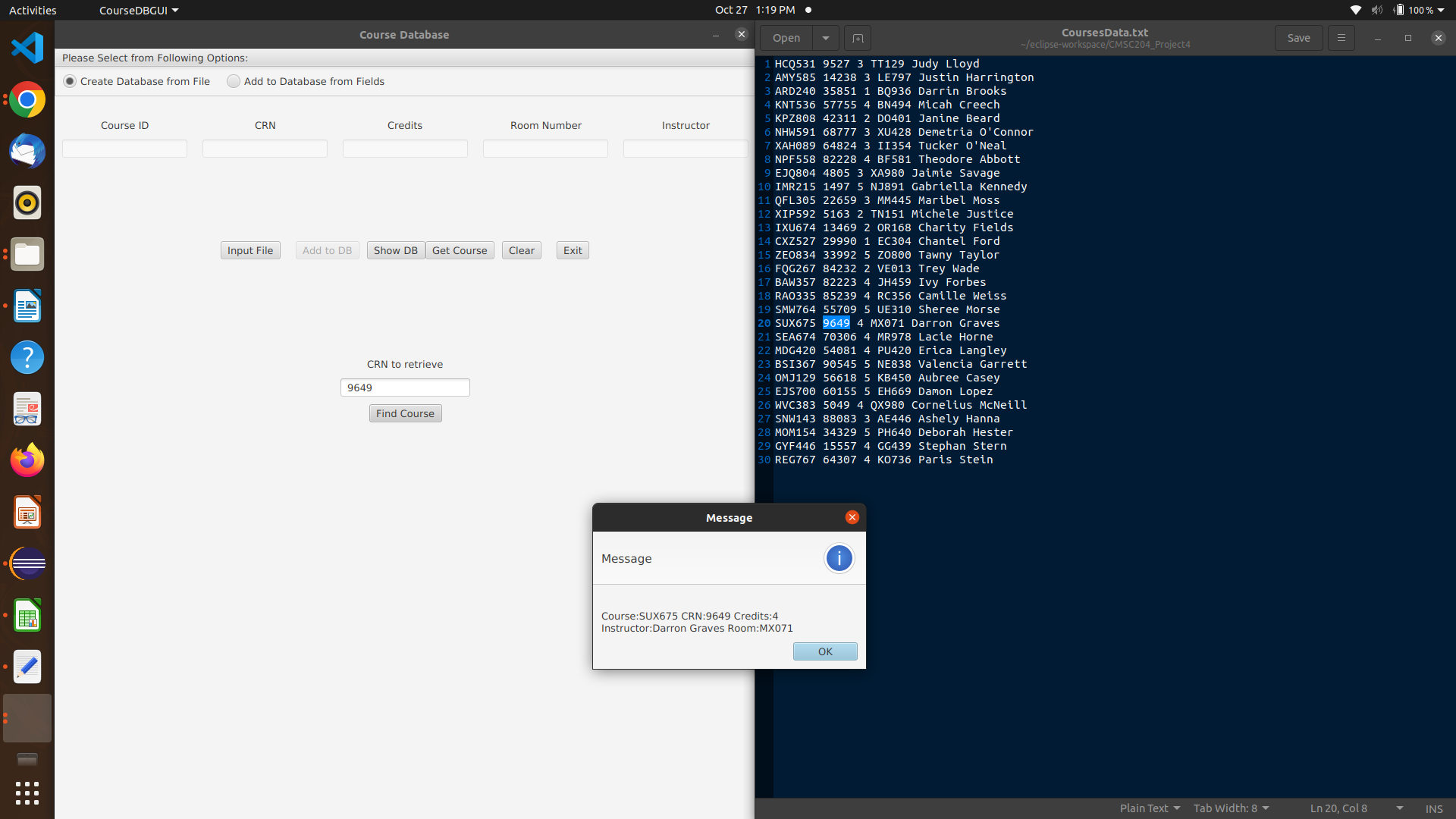
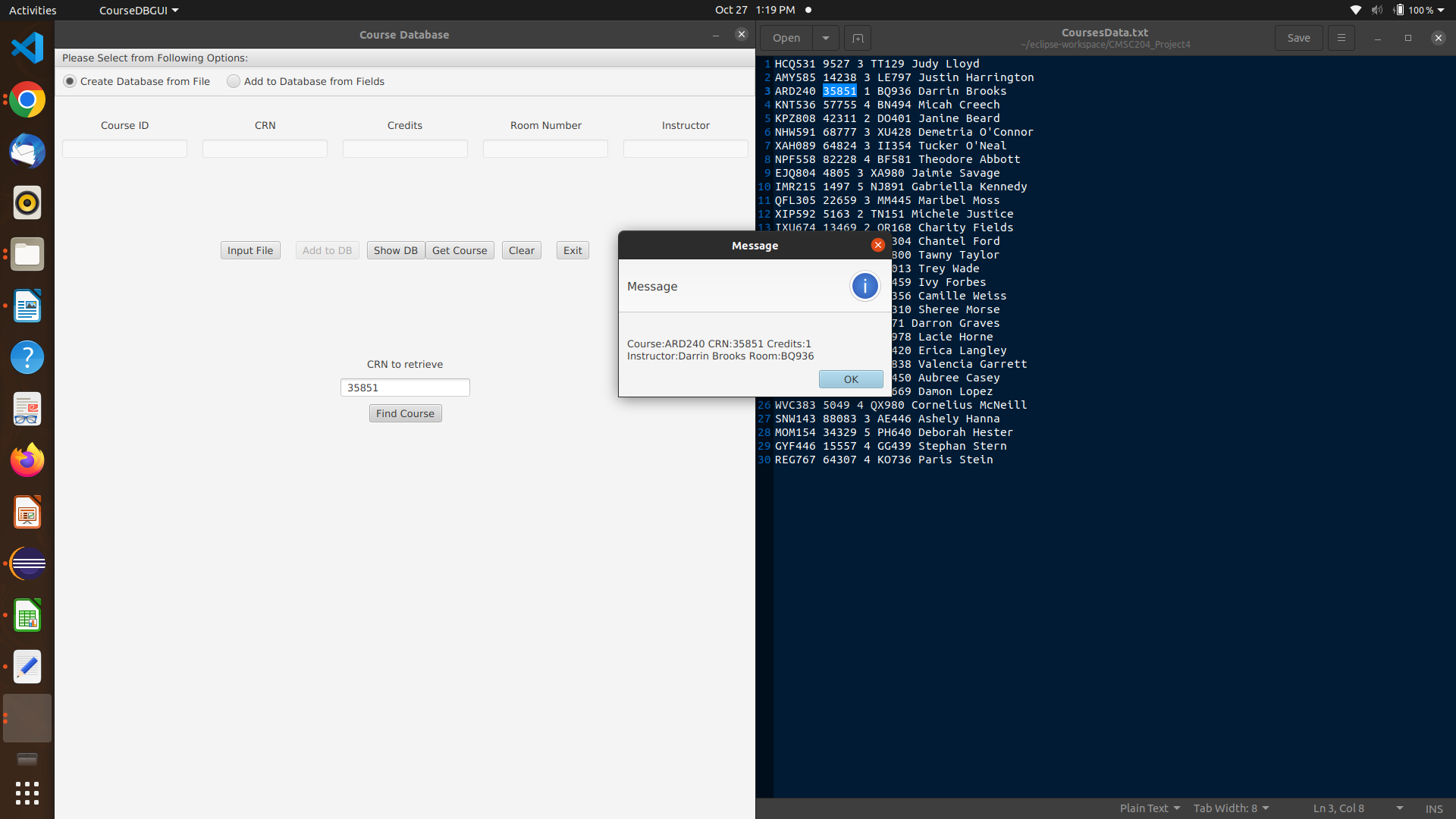
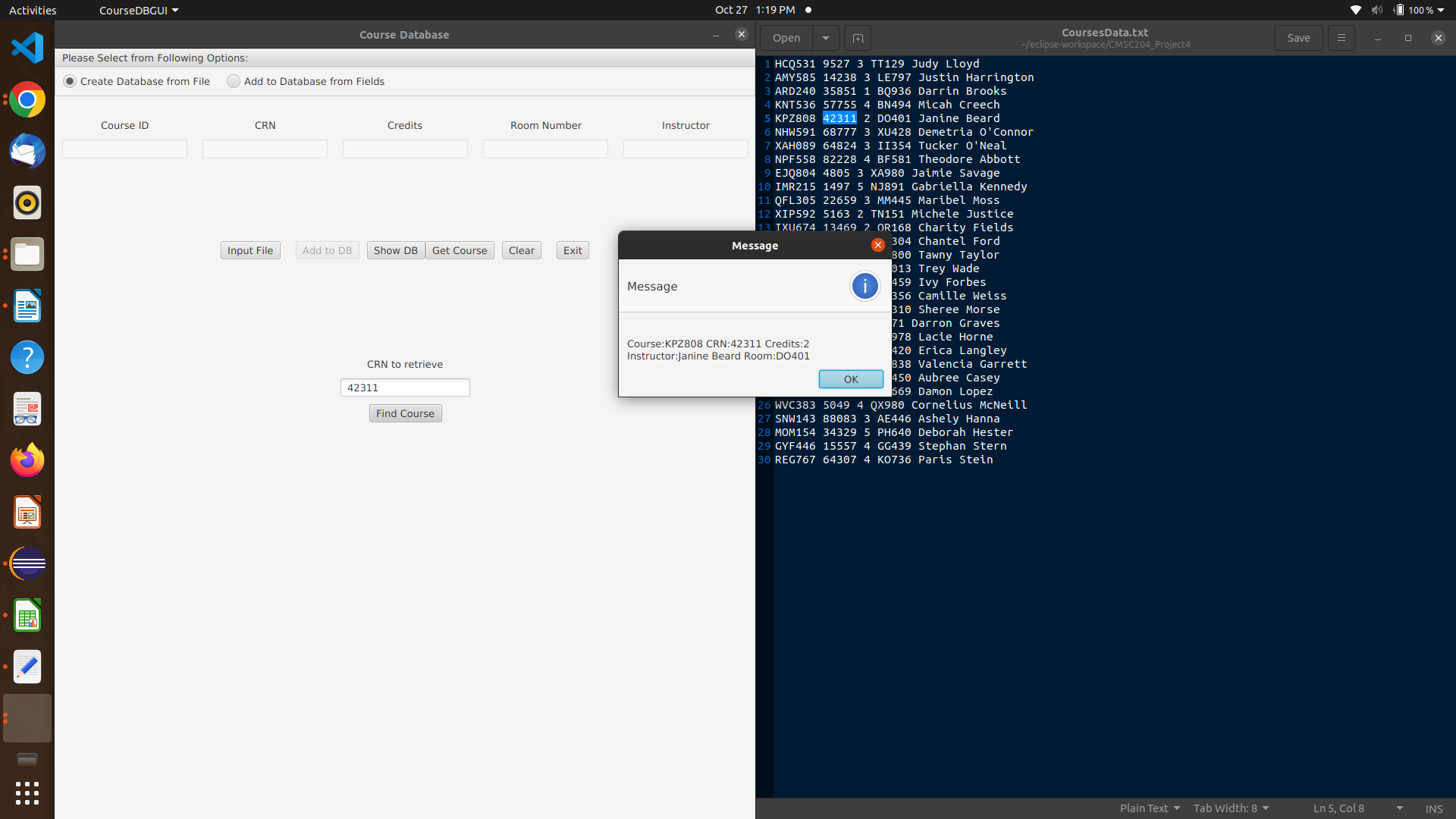
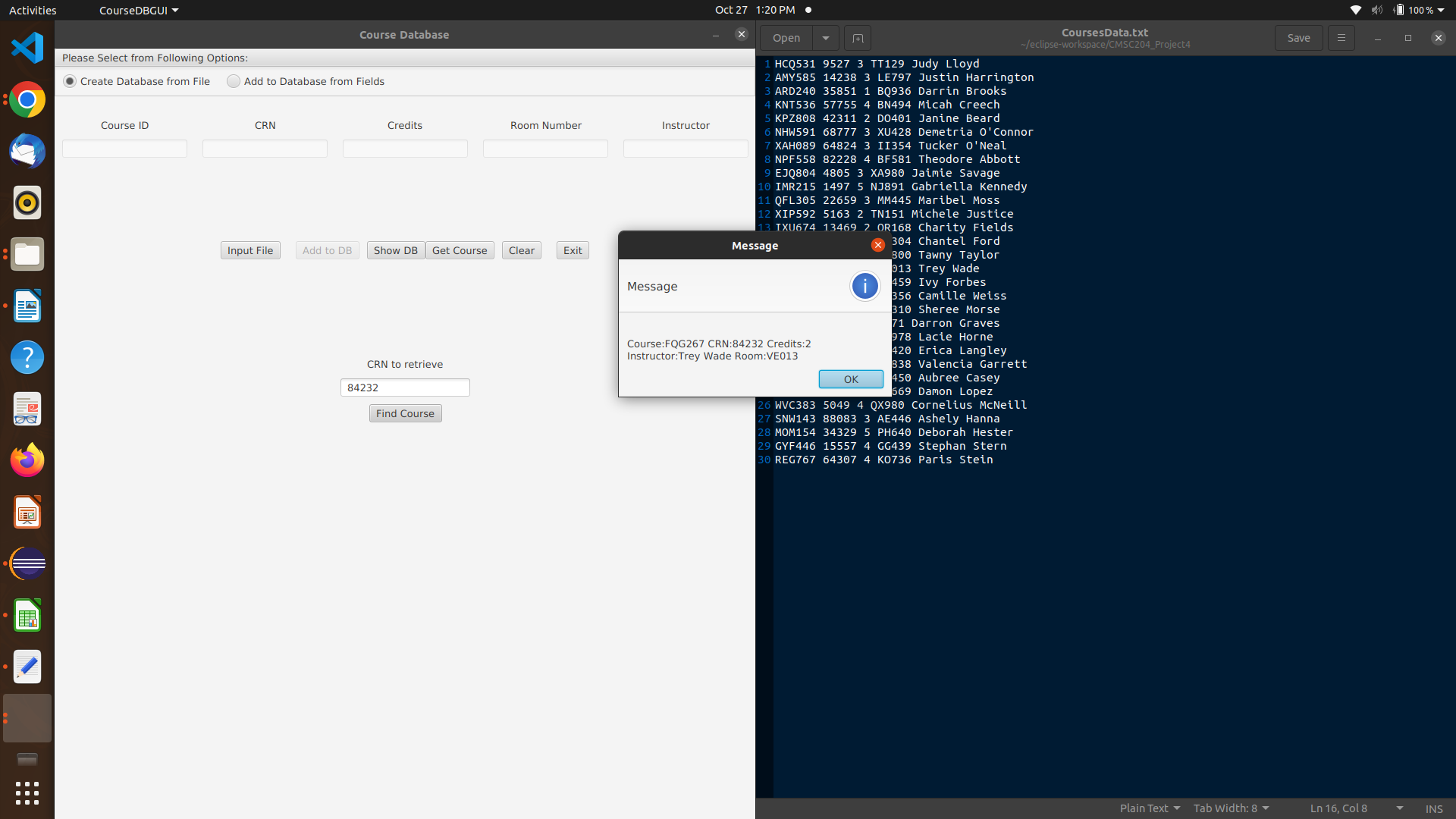
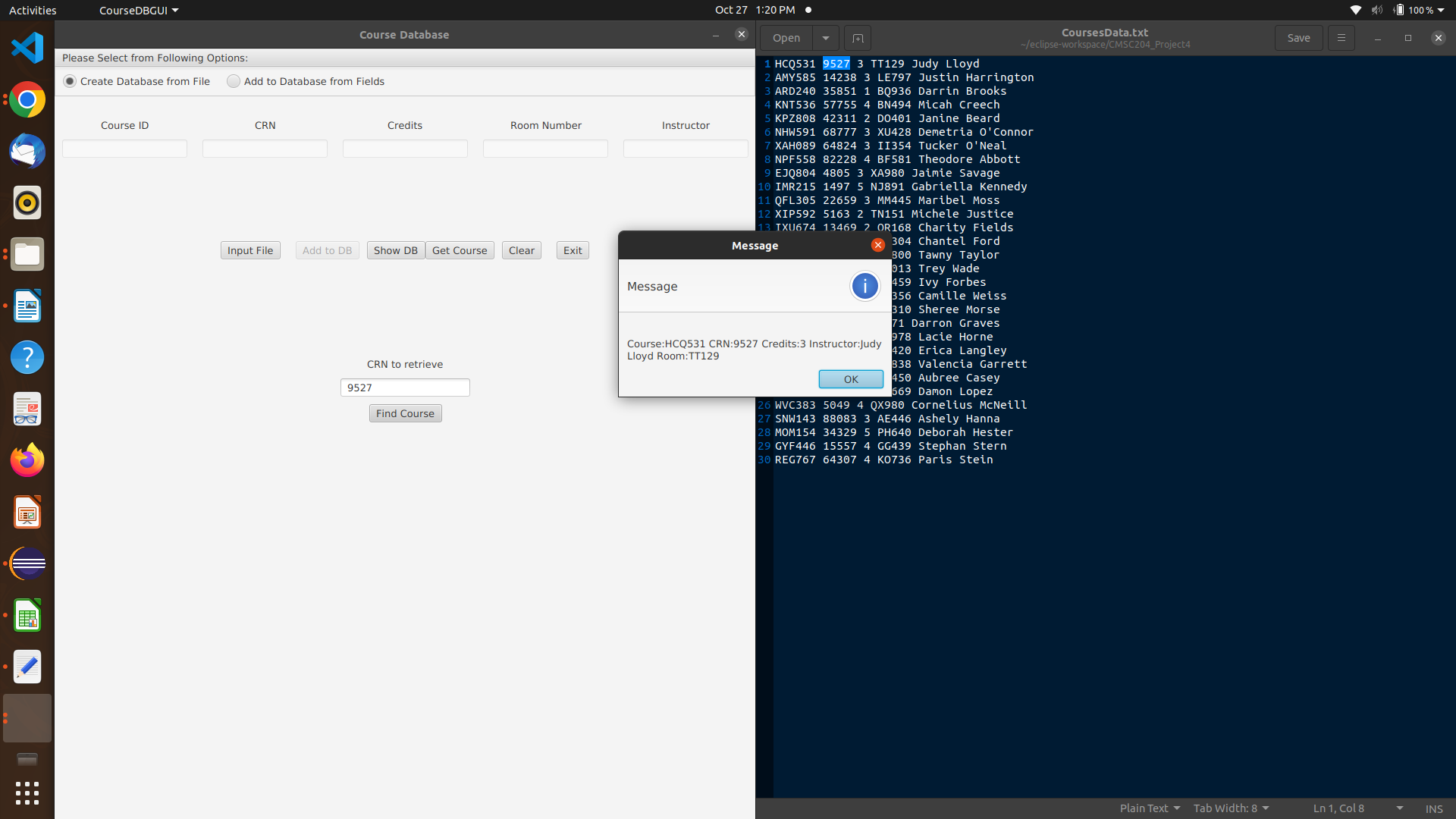
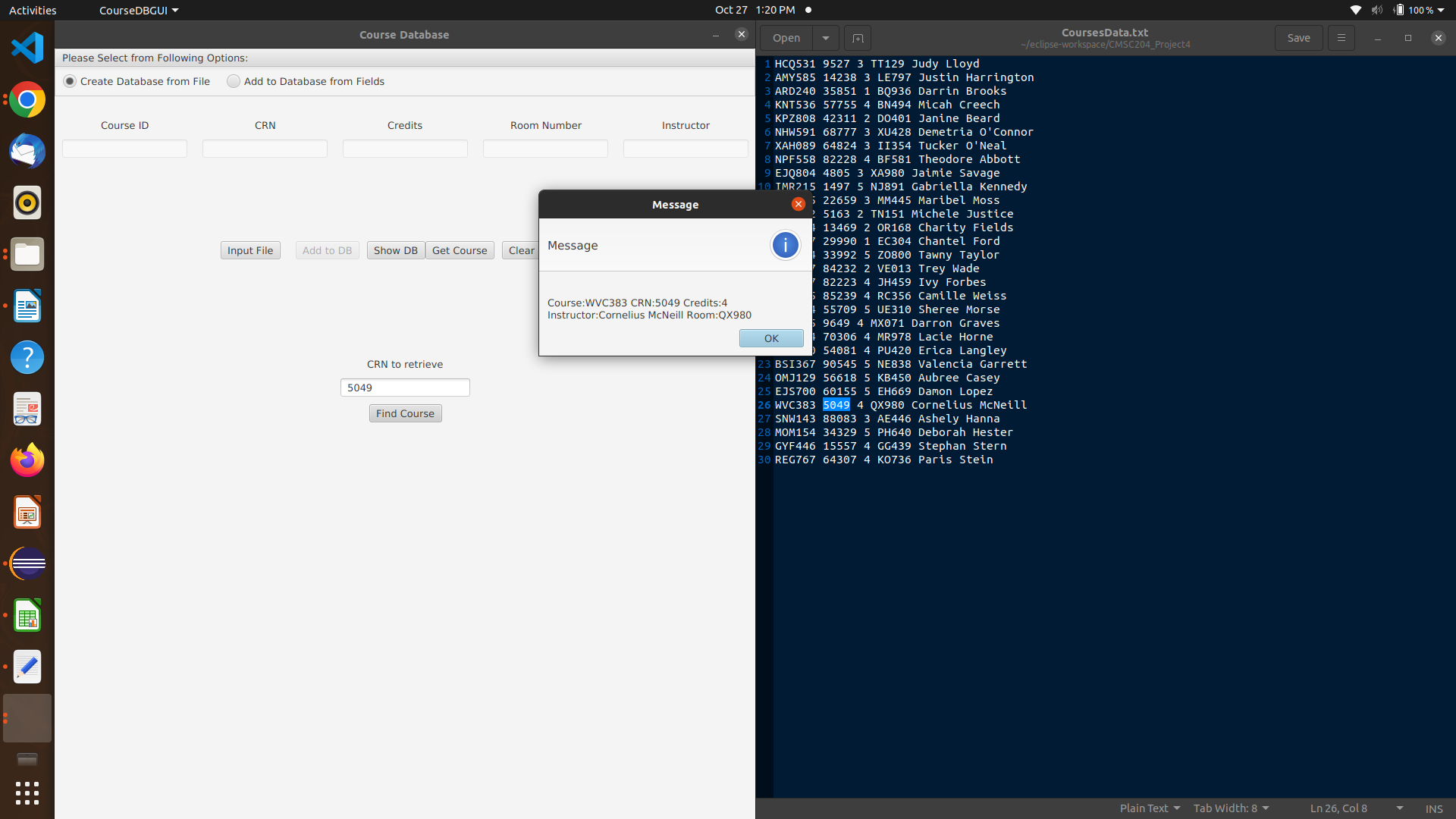


**multiple tests all with the same expected output and actual result:**

case: grab a random CRN from the file and test to if you can find the course

expected: course to be found with correct course information

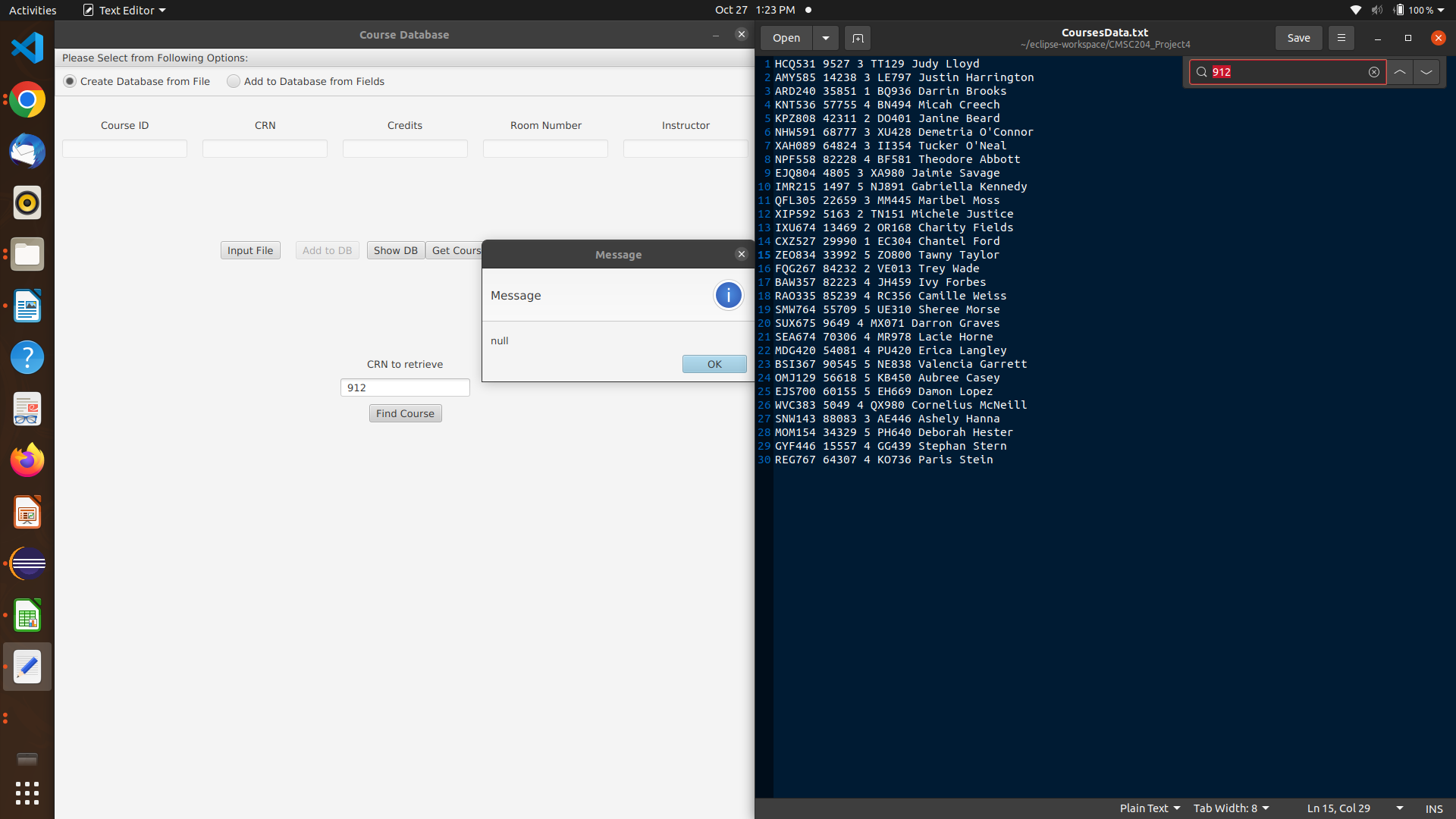
result: as expected



case: search for a crn that is known to not be in the text file

expected: no course info returns or some sort of error depending on the GUI’s implementation

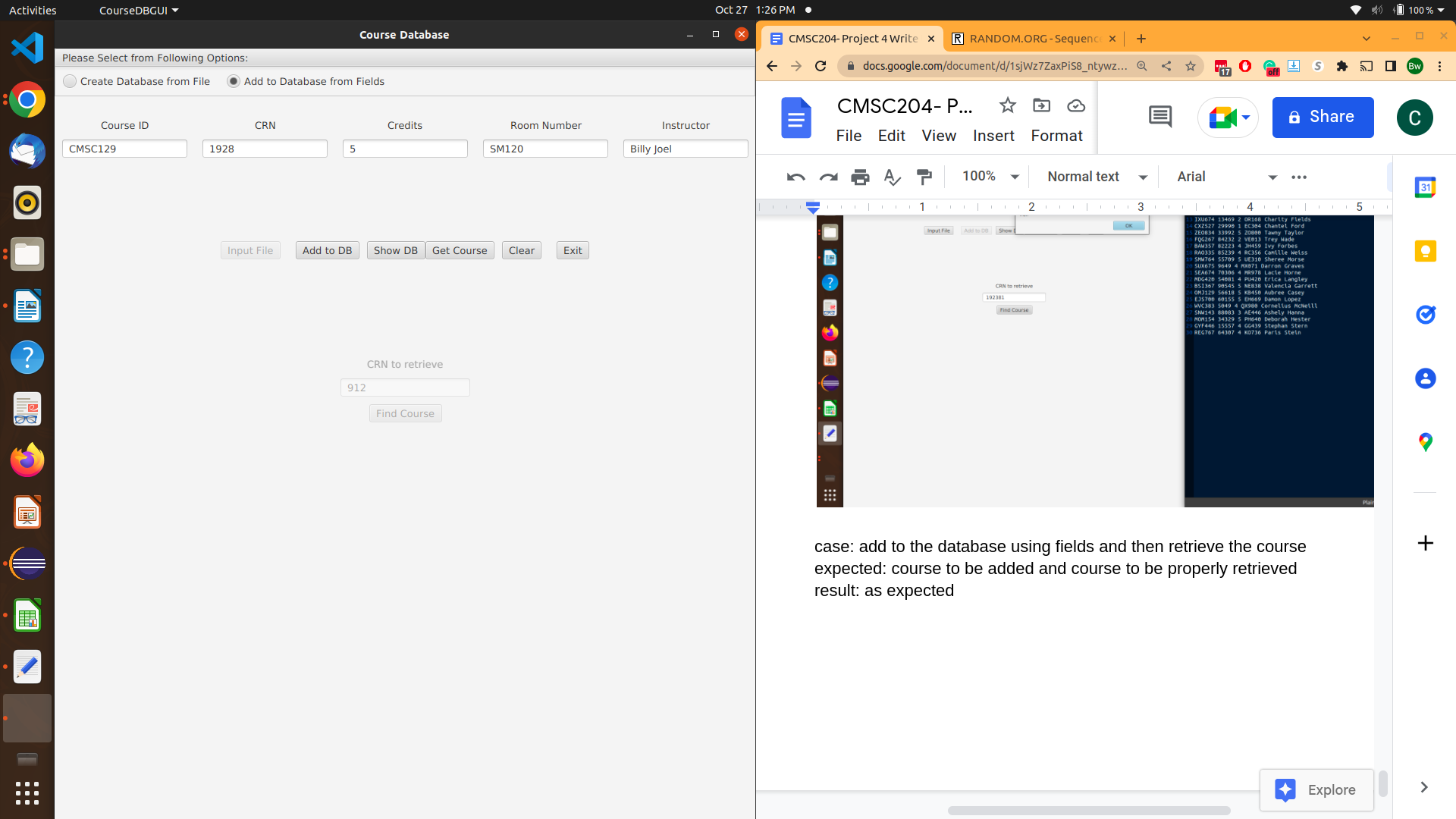
result: as expected



case: add to the database using fields and then retrieve the course

expected: course to be added and course to be properly retrieved

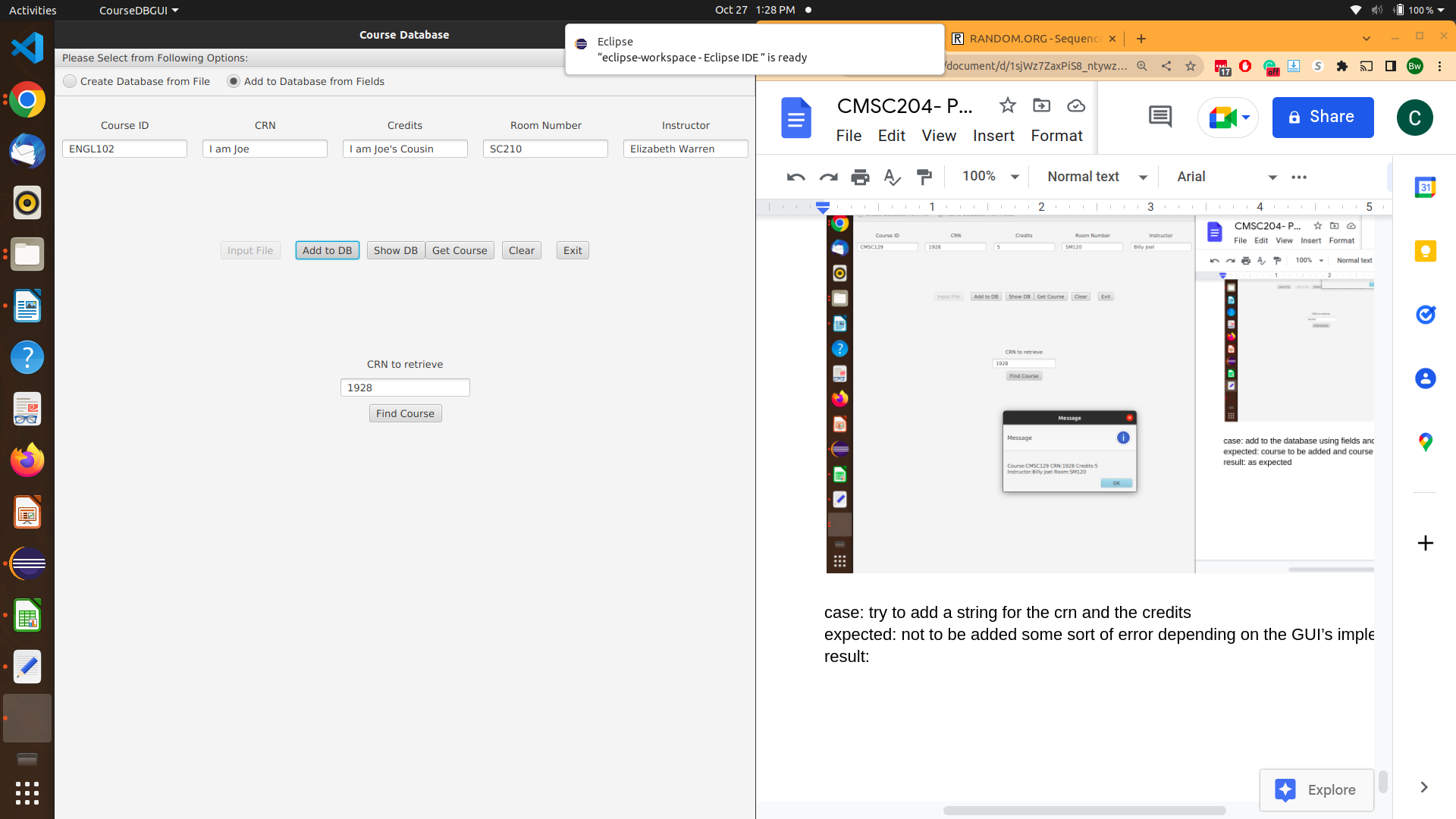
result: as expected



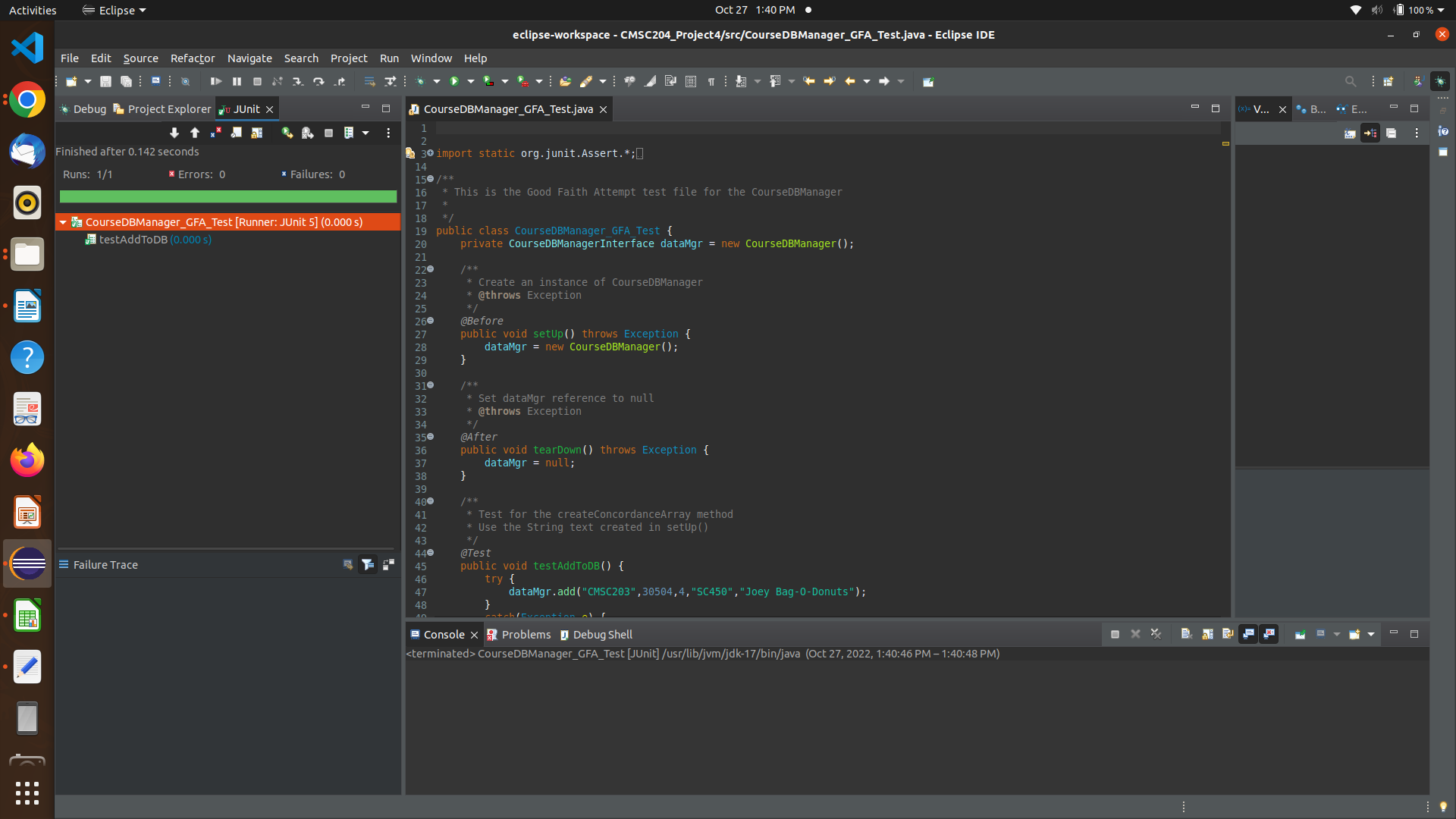
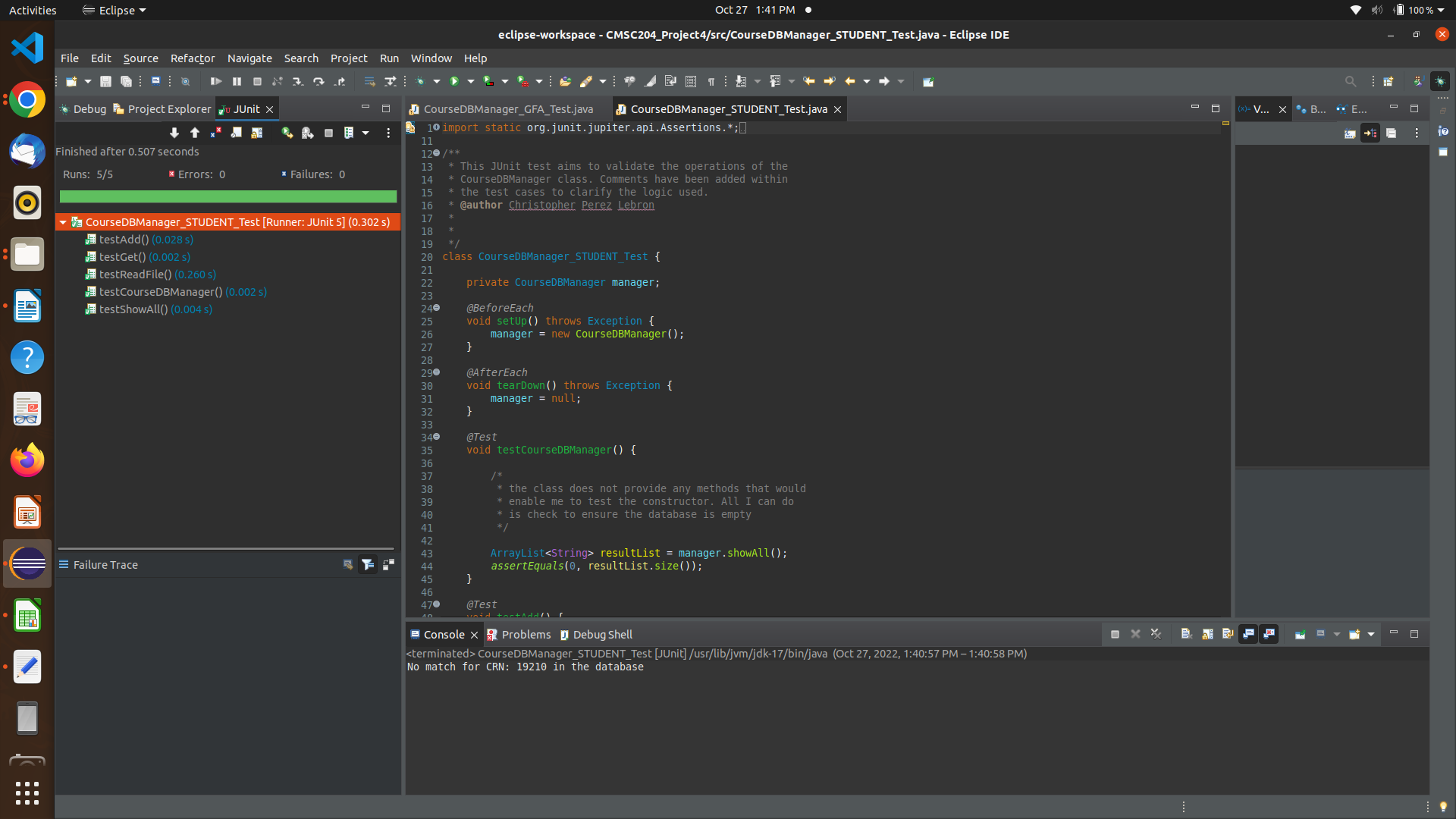
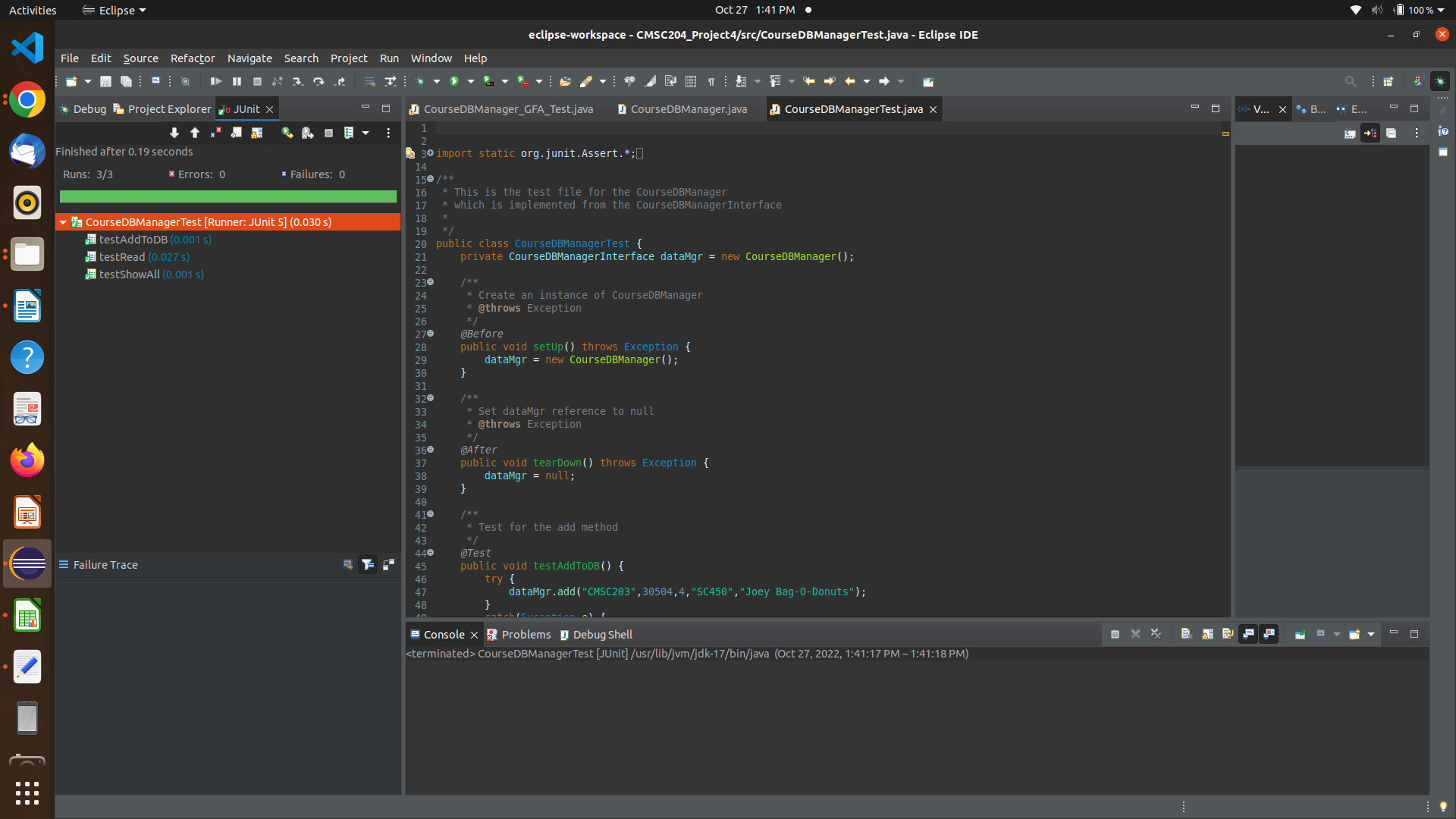
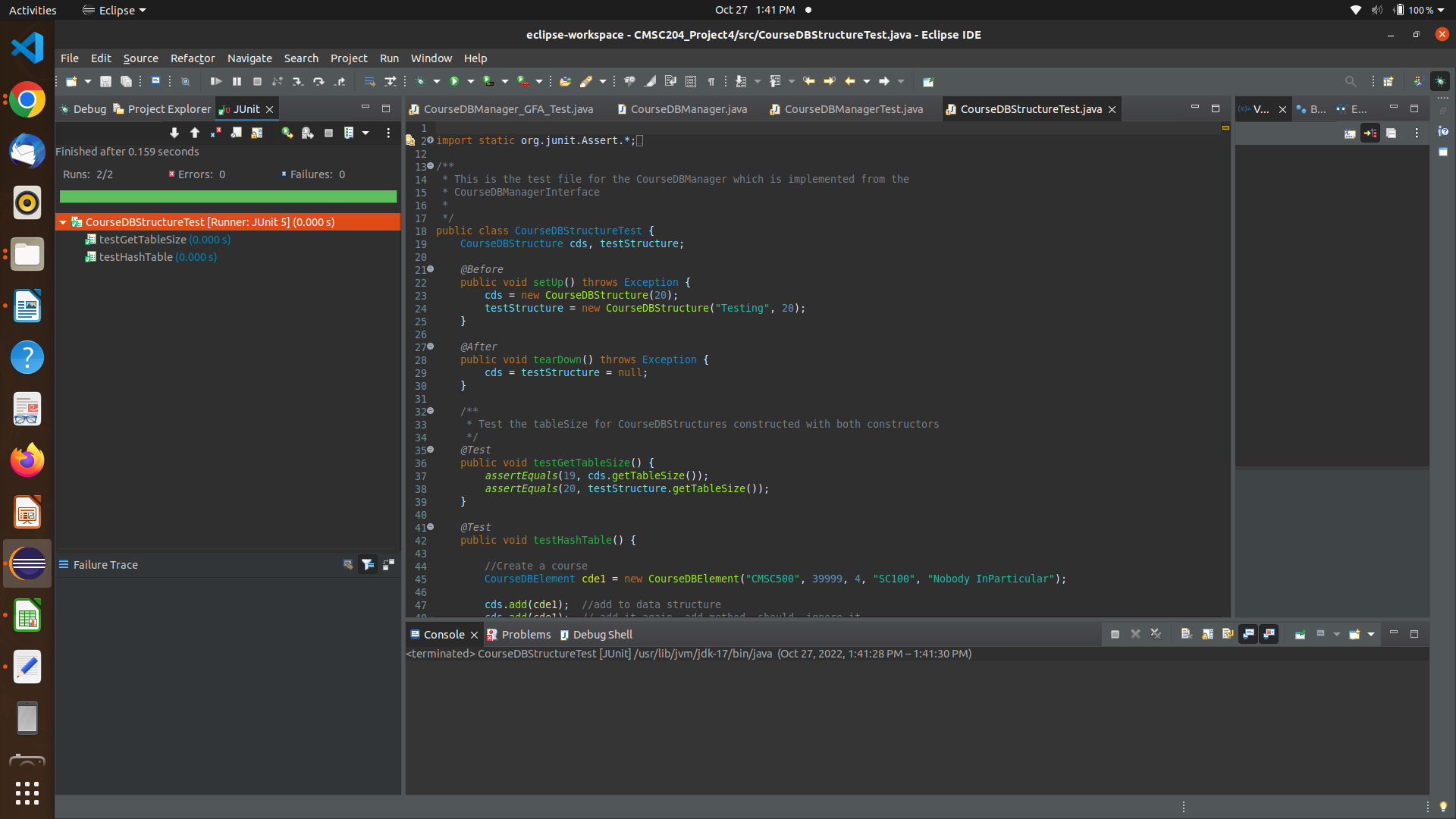
case: try to add a string for the crn and the credits

expected: not to be added some sort of error depending on the GUI’s implementation

result: not exactly what I expected, the whole program crashed. However, that is the GUI’s fault



# JUnit Tests Screenshots



# Github Screenshot

