Programming 2 Coursework

Euan MacKay

### A quick disclaimer:

I wrote my submission entirely using Java 8 Update 231, so if any issues arise, I ~~implore~~ that you test it again in that version - if possible.

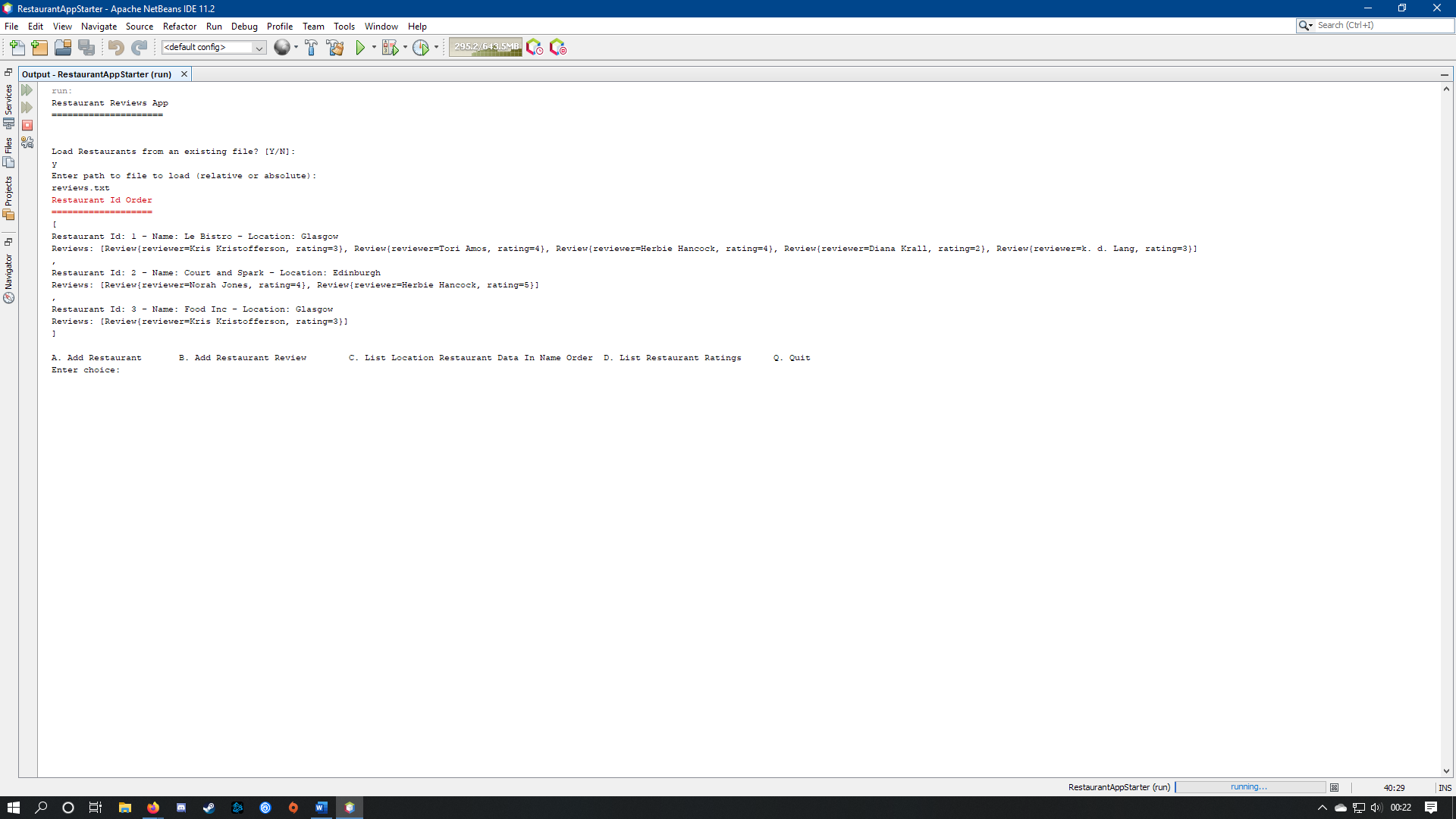
# Increment 1:

Completing the Restaurant and Review classes proved to be rather straightforward; a majority of the work was completed in the starter project. Not a lot of thought was needed for the implementation except for the toString(delimiter) method, which required a quick reanalysis of the specification to determine a use case that would indicate as to what it was used for. I decided to use lambda expressions for the comparators partially because I thought the alternative implementation was horrible to look at and partially because it was an easy extra mark.

For the Repository class I decided to use a List as my collection type seeing as a List was used already for the reviewsCollection to fulfill a similar role. However, I generified the Repository class by introducing type specification and an abstract RepositoryObject. The type is required to be an extension of RepositoryObject, so this way I can guarantee that the type being passed in has a getItem(int id) function. As such, I had to update the Restaurant class and RestaurantController class to extend RepositoryObject and specify the Repository type respectively. For the load() and store() functions I simply created a new DAOImp object and ran the respective method on that, passing in the file to load.

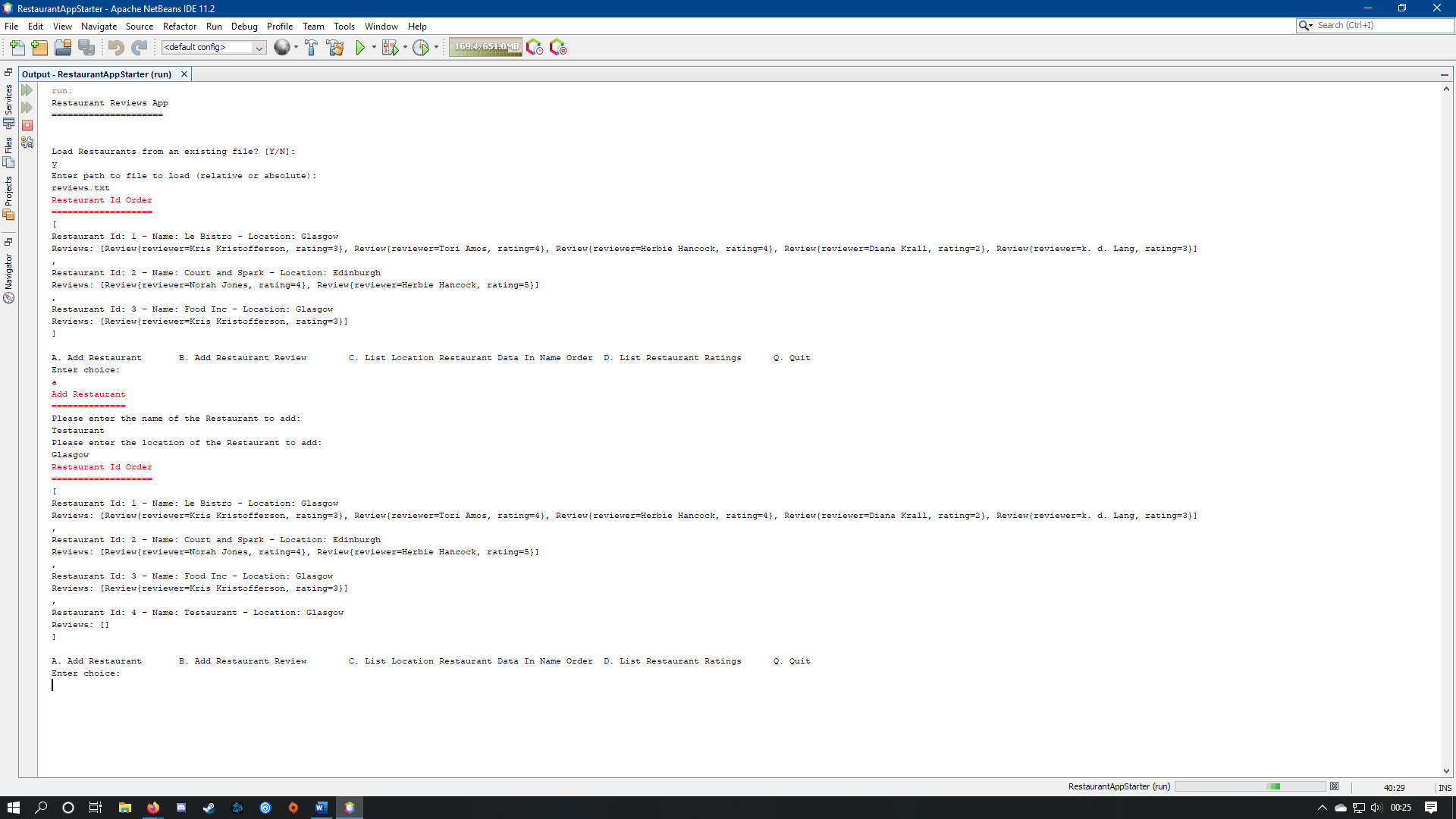
When it came to writing the implementation for the DAO I used a text-based implementation so that I could take advantage of the toString(delimiter) and my IDE would stop shouting at me because it was unused. Taking advantage of FileWriter and the PrintWriter objects, the store() method takes each entry in the passed repository and prints the toString(delimiter)’s return value to the specified file. The load() function attempts to do the inverse with FileReader and BufferedReader objects; reading in a line, splitting it with the delimiter we used in store() and stripping the quotes from data as required - before creating a Restaurant object to return to our repository.

In the RestaurantController class I used the supplied InputHelper class to read an input character from the user, converted the result to lowercase (if not already) and queried the user further for a file to load had they chosen to do so. I then had the RestaurantController create a Repository object to store the data based on the user’s inputs. The listRestaurantDataInIdOrder() method uses the Comparators I created earlier in the Restaurant class to sort the repository and prints its contents.

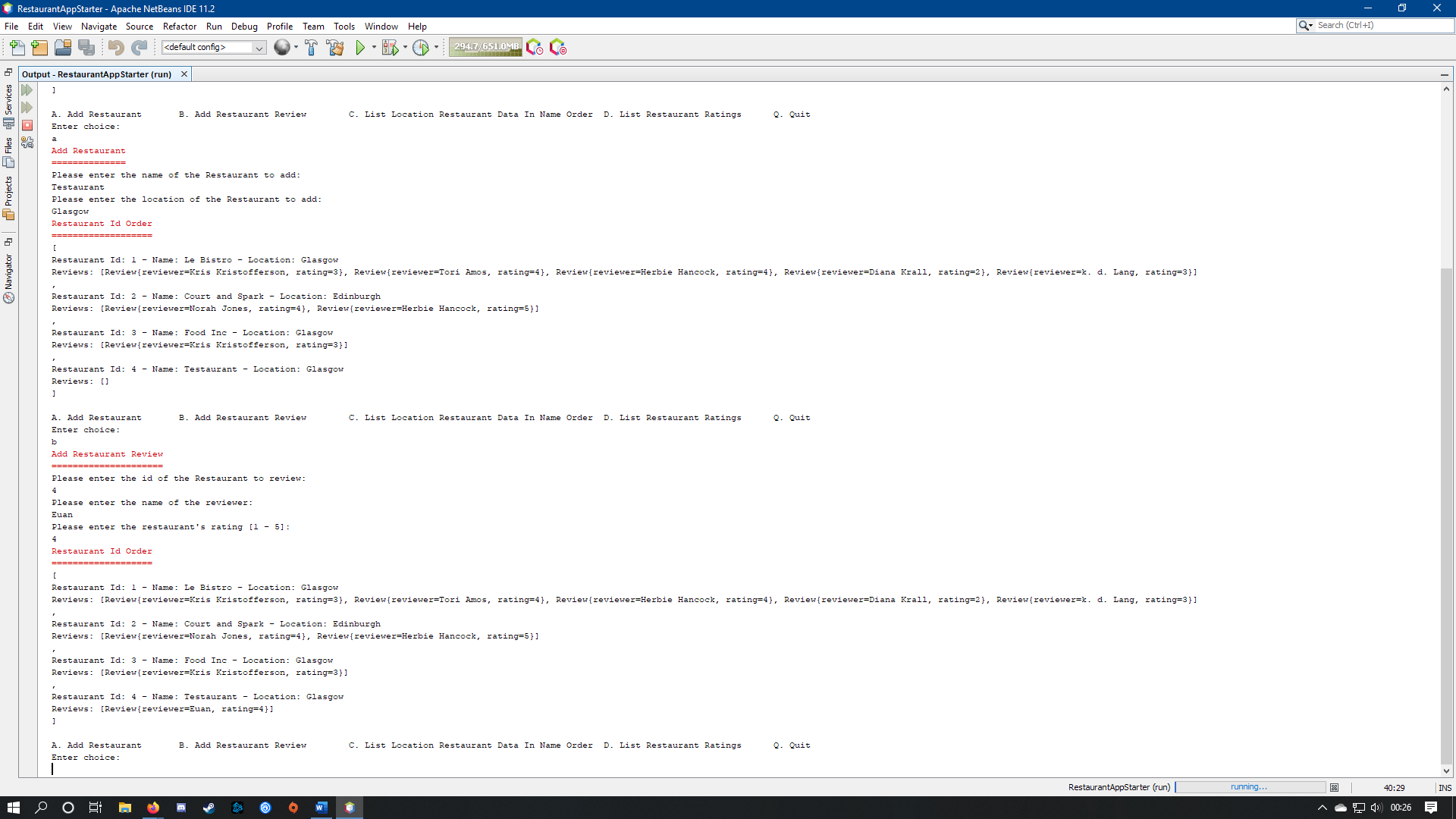


# Increment 2:

Increment 2 turned out to be quick and easy to implement. In the RestaurantController class I created a method to ask the user the name and location of a restaurant they would like added to the application. It then takes that data and creates a new Restaurant object with it that gets added to our repository.

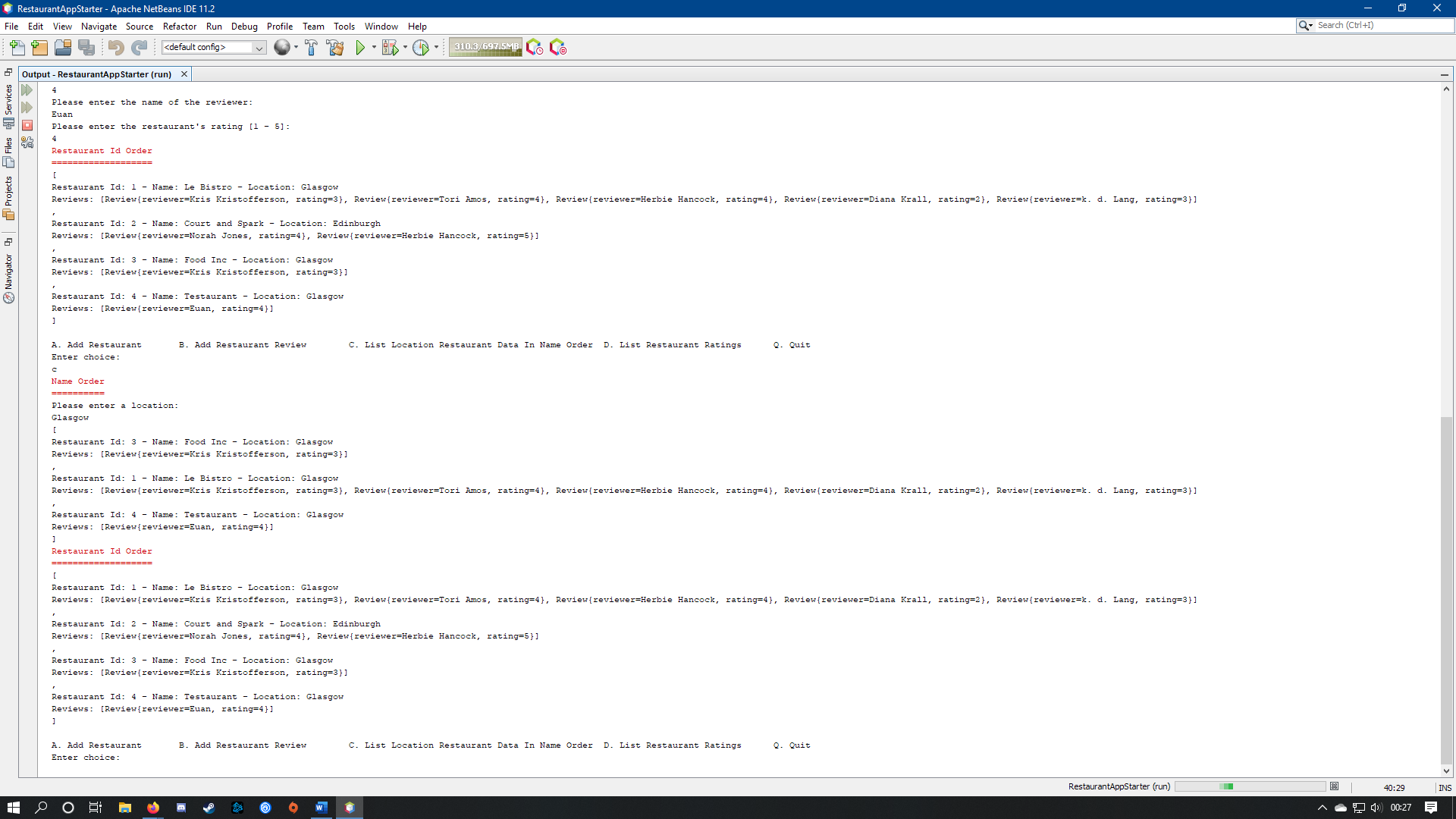


# Increment 3:

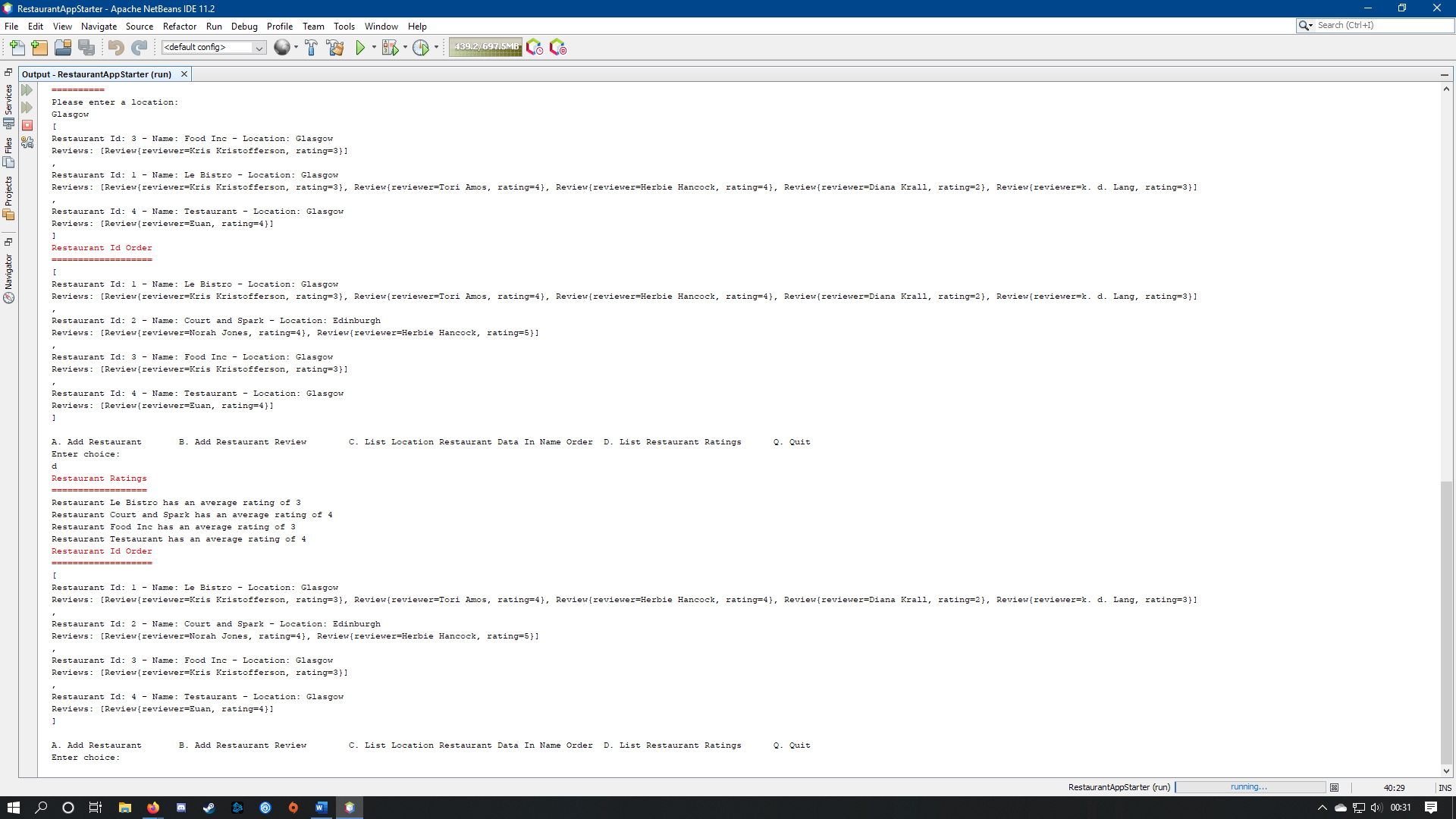
Like before, Increment 3 proved little challenge to implement. A new method was created that prompts the user for an id of a restaurant present in our repository, to which they would like to add a review. It then proceeds to ask the user for the name of the reviewer and their rating, with which a new Review object is created and added to the reviewsCollection of the specified Restaurant object in the repository.

# Increment 4:

Like the above, this was short and simple to implement. I used the provided InputHelper yet again in a new RestaurantController method to prompt the user for a location, with which it iterates through the Restuarants saved in its repository and copies into a new List object. Said List object is sorted with the respective Comparator in the Restaurant class and displayed on-screen.

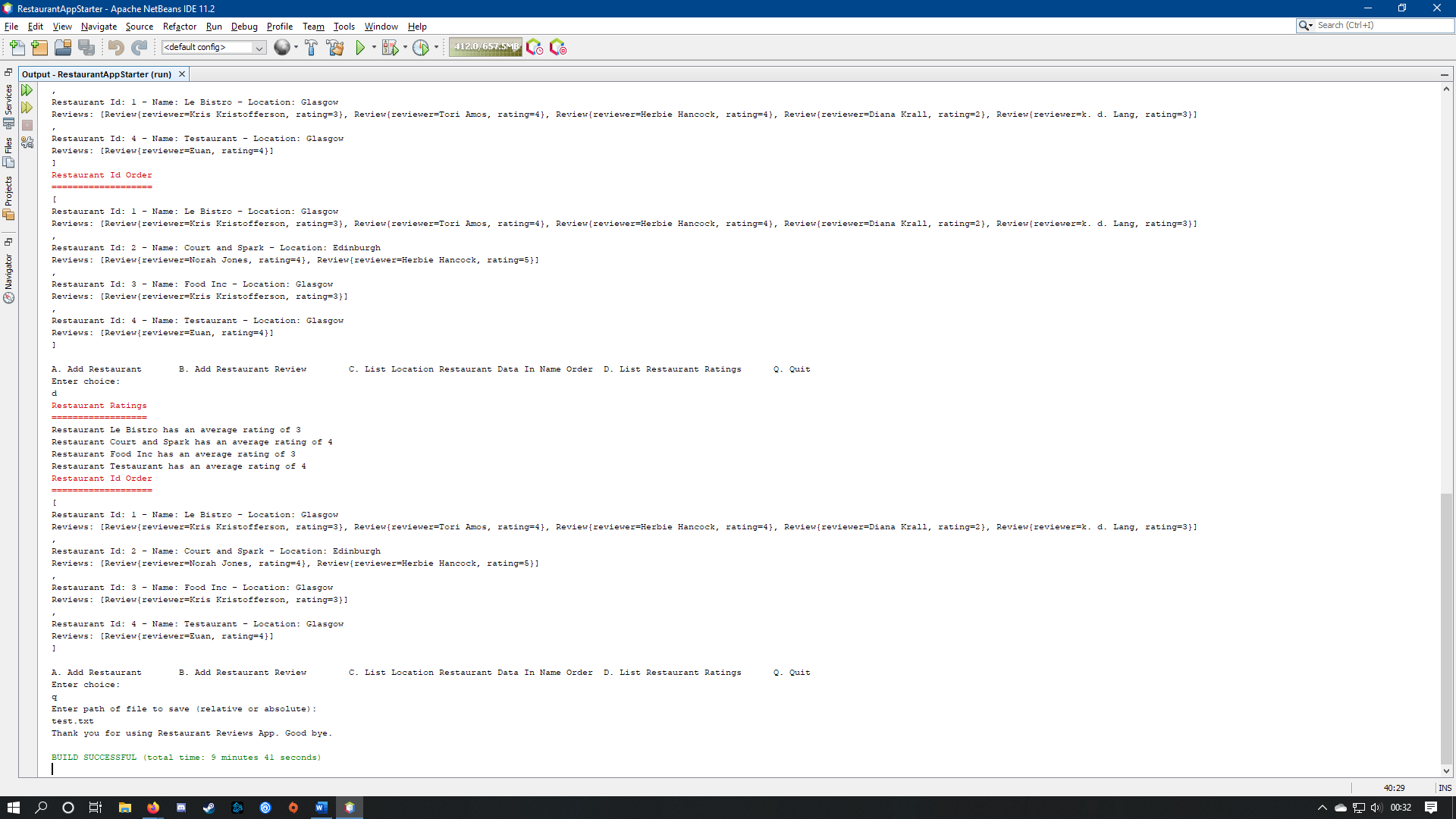


# Increment 5:

Slightly more challenging than its predecessors, I created a new method inside RestaurantController using a nested loop. In the outer loop, I iterated through each Restaurant in the repository and created a local variable that would save the average rating for that loop, run the inner loop, and divide the variable by the number of loops ran in the second before displaying the result. In the inner loop, I iterated through the Reviews stored in each Restaurant object, and sum the rating using the variable created earlier.

# Increment 6:

I had to take some creative liberties when it came to Increment 6, as the specification does not once mention where the store function in the repository should be used. I decided that the application would ask the user for a location to save the repository to whenever they were finished.

I did this by adding an InputHelper object that queries the user for a file to save to in the quit case of run()’s switch, before calling the repository’s store function with the result.