Project6 Write-up

I started by understanding the project requirements and breaking the work into smaller, more manageable parts. The design made sure each class had a clear and specific job.

The **Town** class represents individual towns in the graph. It includes the town name and methods to check if two towns are the same or to sort them. The **Road** class handles connections between towns. It keeps track of the two towns it connects, the road name, and the distance. Since roads work both ways, I made sure the class treated connections from A to B the same as B to A.

The **Graph** class manages the main graph structure using an adjacency list. It handles adding, removing, and checking towns and roads. The **TownGraphManager** makes the graph easier to use by providing methods for reading files, finding paths, and interacting with the GUI.

To find the shortest path between towns, I used Dijkstra's algorithm. It required careful tracking of distances and previous towns to make sure the paths were correct. I also made sure the program handled cases where towns weren't connected by returning an empty path.

**Key Features**

The project includes efficient shortest path computation using Dijkstra's algorithm. It also handles edge cases, like when towns are not connected, and gives clear error messages when something goes wrong. The modular design makes it easy to test, fix, and expand the program later.

**Testing**

**JUnit Tests**

I wrote tests to make sure everything worked as expected:

* **TownGraphManagerTest:** Checked that towns and roads could be added, deleted, and queried. Also tested pathfinding.
* **GraphTest:** Made sure the graph correctly handled edges, shortest paths, and bidirectional connections.

I also created additional custom tests:

* **TownGraphManager\_STUDENT\_Test:** Used a different dataset to test the same methods, ensuring the program worked with new inputs.
* **Graph\_STUDENT\_Test:** Tested graph methods using unique data.
* **Road\_STUDENT\_Test:** Checked that roads were treated as equal if they connected the same towns, regardless of direction.
* **Town\_STUDENT\_Test:** Made sure towns were compared and sorted correctly.

**Test Results**

The tests covered adding and removing roads, finding paths, and handling disconnected graphs. All outputs matched what I expected after fixing some initial issues. For example, pathfinding now works perfectly, and disconnected graphs return an empty list instead of causing errors.

**JavaFX Runs**

Using the JavaFX GUI, I tested adding and removing towns and roads, as well as finding paths between towns. The GUI showed clear error messages for invalid inputs and worked smoothly. Screenshots show examples of adding roads, finding paths, and displaying the graph.

**Fixed Issues**

During development, I ran into several problems:

* **Path Backtracking:** At first, paths were not constructed correctly. I fixed this by keeping track of the previous town for each step in Dijkstra's algorithm.
* **Disconnected Graphs:** The program didn’t handle disconnected towns well. I updated it to return an empty path when no connection exists.
* **Undirected Edges:** The getEdge method only checked one direction, so I updated it to consider both directions.
* **Road Equality:** The Road class was updated to ensure that roads from A to B were treated the same as roads from B to A.
* **File Handling:** Added proper error messages for missing or unreadable files.

**Learning Experience**

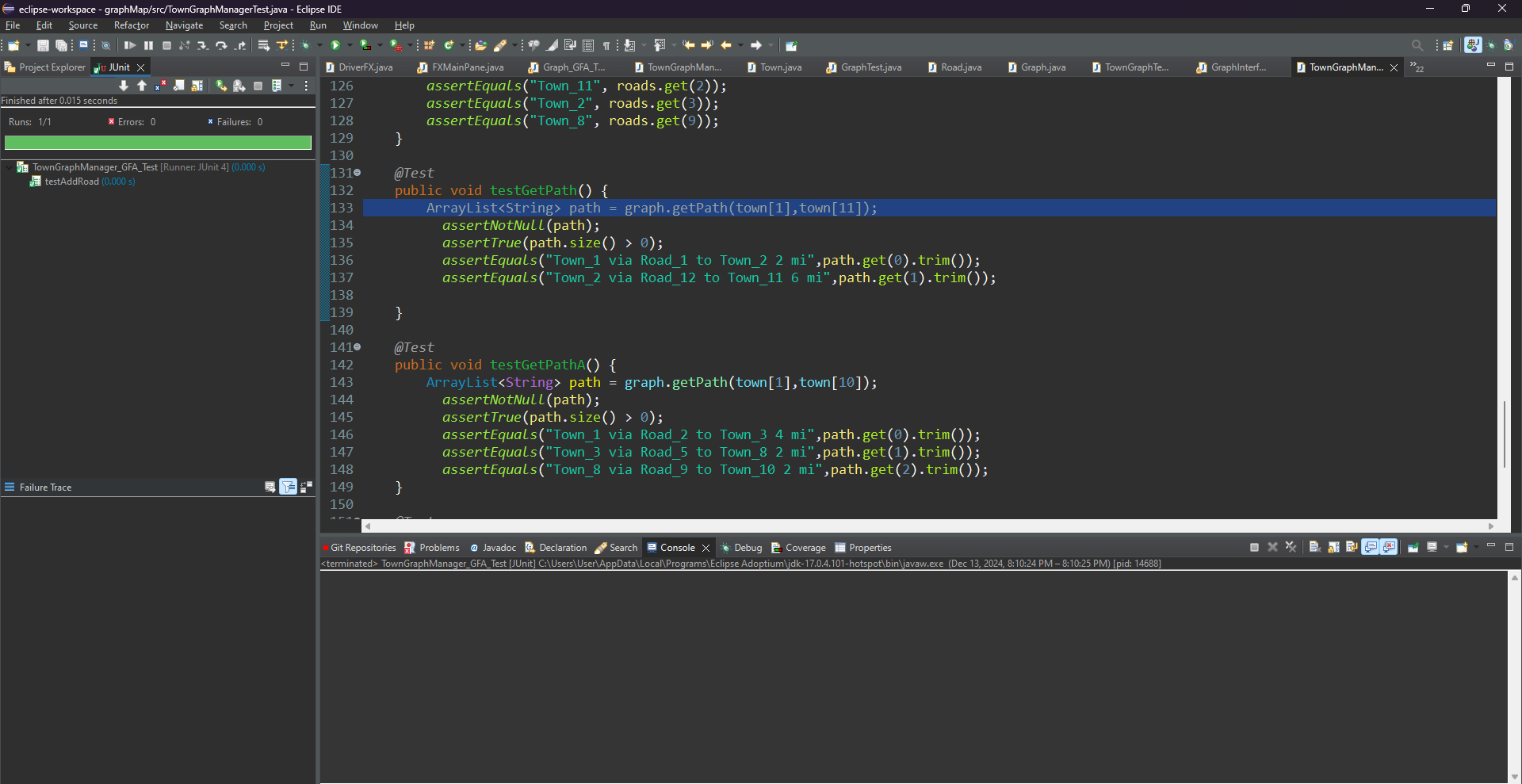
This project taught me a lot about graph algorithms, especially Dijkstra's algorithm. I learned how to handle tricky cases, like disconnected graphs, and how to write code that is easy to test and debug. Writing JUnit tests helped me find and fix issues quickly. I also learned how to connect the backend to the GUI so that everything worked together.

**Assumptions**

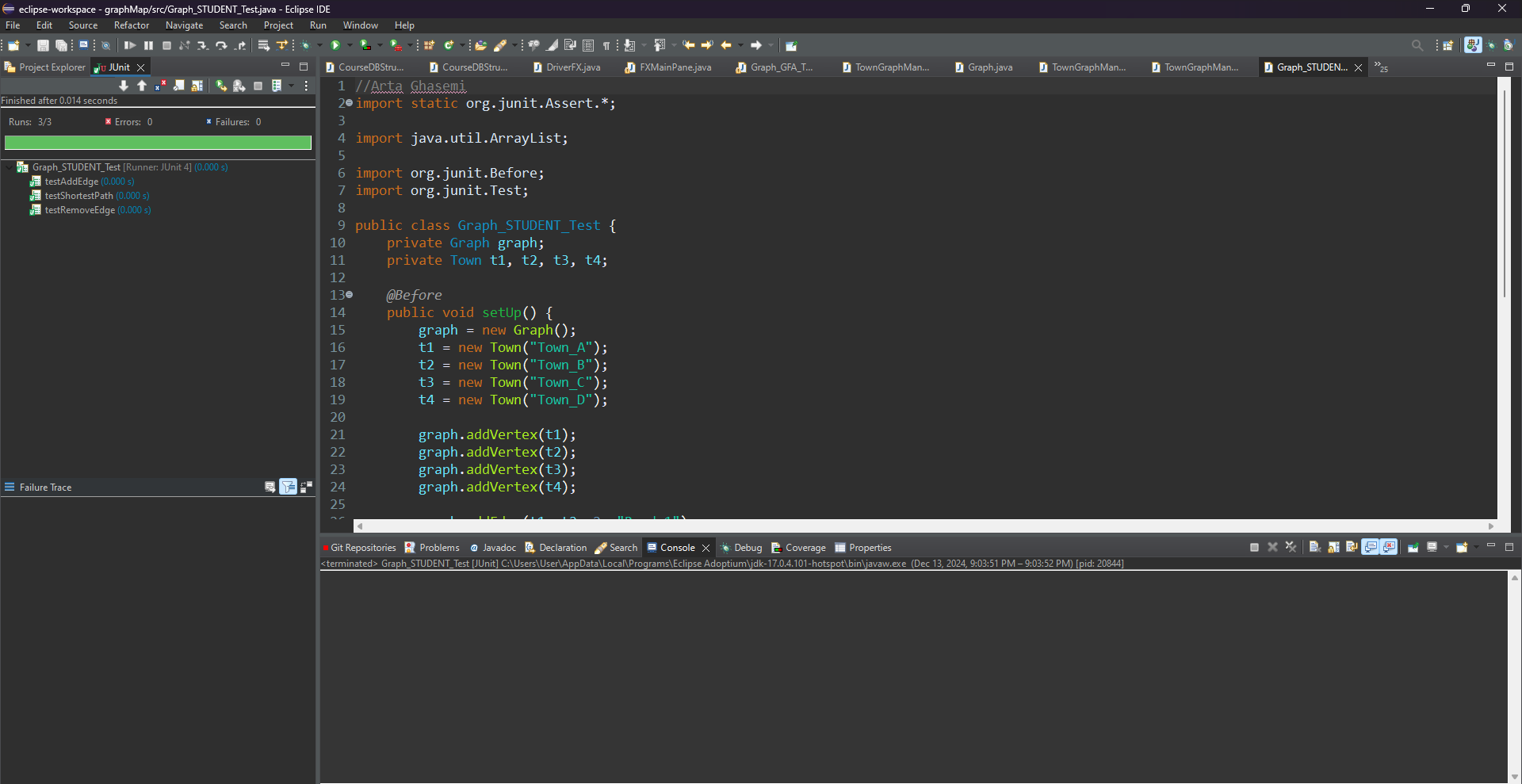
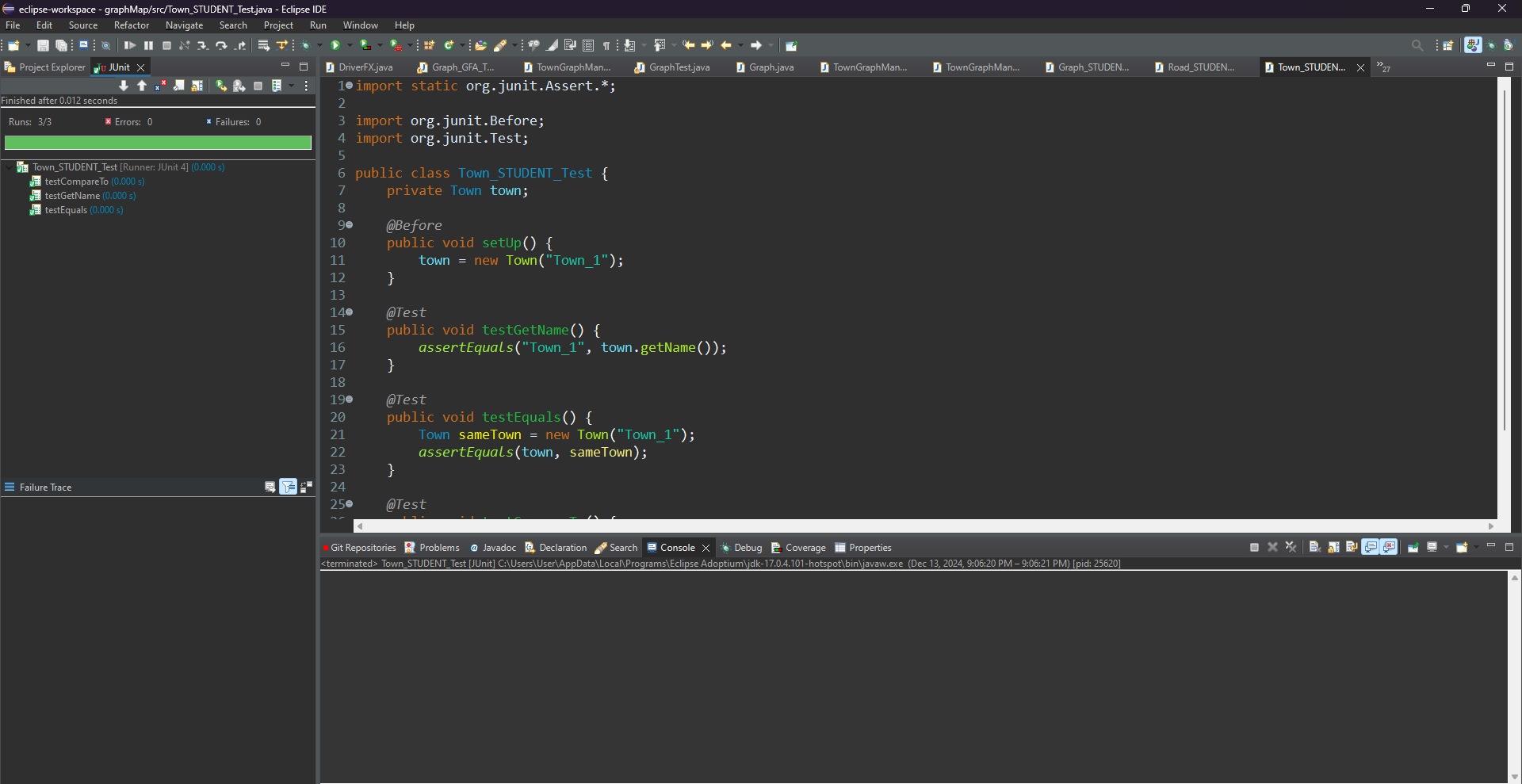
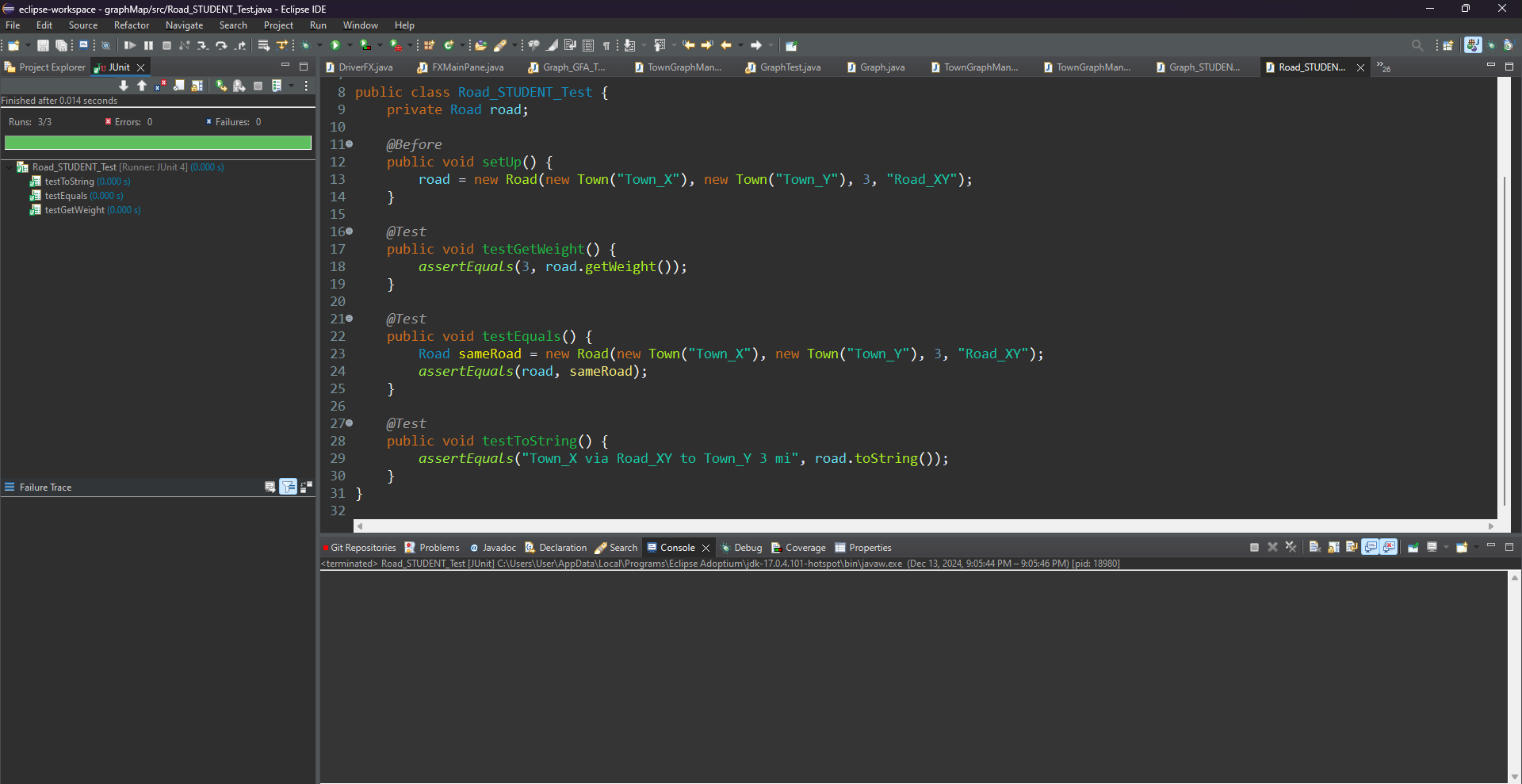
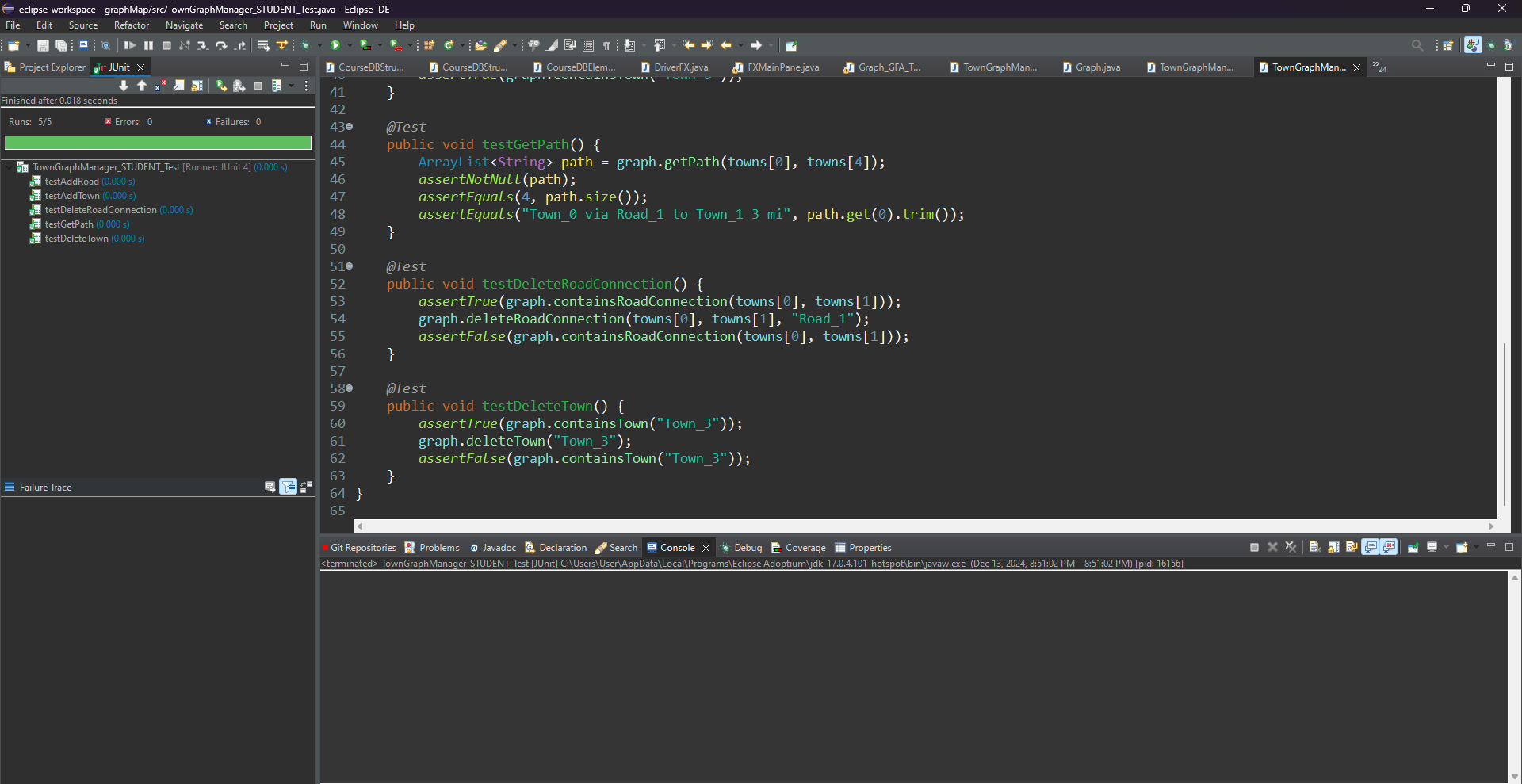
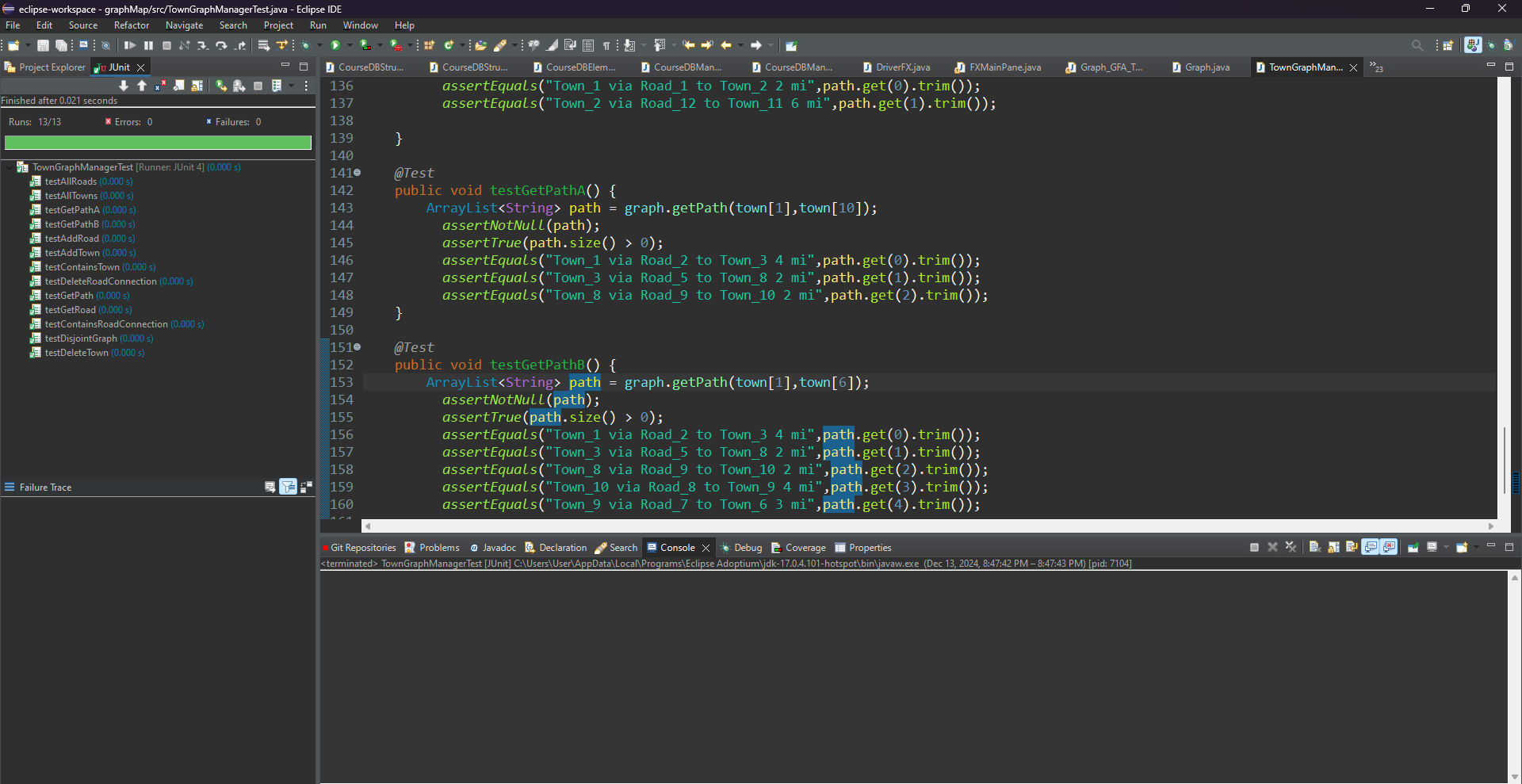
* The input file follows the example format provided.
* Town names are unique.
* Roads are undirected and have non-negative distances.

**Submission Details**

A screen shot of a computer

Description automatically generatedThe project includes all Java files, JUnit tests, and screenshots of JavaFX runs. The program passes all tests and is designed to be easy to expand or maintain in the future.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

