Names: Madison Betz and Kaiden Pollesch

Date: 5/6/2025

Assignment Title: Lab 14 Triangle Trouble Revisited

Repository Link: https://github.com/msoe-SWE2721/2025-lab-14-triangle-revisited-121_betz_pollesch_lab14_2025.git

Introduction

What are you trying to accomplish with this lab?

In this lab, our goal was to design, implement, and thoroughly test a software system capable of evaluating triangle definitions. We were tasked with extending the work from Lab 1 by creating both the triangle side lengths from user input or files, verifying if the sides could form a valid triangle, computing area and perimeter, and determining the type of triangle (equilateral, isosceles, or scalene). The purpose of this lab was not only to revisit triangle classification but also to work more with a range of testing methodologies to ensure code correctness and robustness.

Testing Discussion

For testing, we applied multiple techniques to ensure our code covered a wide range of scenarios:

- Triangle Class: we used boundary value analysis (BVA) and modified condition/decision coverage (MCDC). Since this class contains logic for triangle validity and type detection, we focused on edge cases (sides with length 0 or just violating the triangle inequality). We also tested known triangle types of confirm correctness.
- NumericParserClass: we employed exception- based testing here. We tested valid integers, strings containing non-numeric characters, empty strings, and null inputs to ensure that the custom exception NumericParseException was triggered appropriately.
- TriangleFactory Class: we used input redirection and mocking scanner inputs to simulate reading from files or user input. We verified the aggregate functions like total area and count, as well as string output formatting.

We organized our test cases using Junit in separate test files corresponding to each class. Each test case was clearly labeled and grouped by method being tested. We also included comments to indicate what aspect (boundary, invalid input) each test was targeting.

Things Gone Right / Things Gone Wrong

Things that went right in this lab were creating the tests. We were able to use tools and other testing functions that we learned about in class to make sure that our tests would find bugs and errors in the system. It also helped that we have done some previous work in this lab, so we have a greater understanding of how the application functions.

Things that went wrong in this lab were getting our tests to compile. We have frequently had issues with output paths not being found and the tests would not run. This may be an issue with Gradle, but it causes lots of problems. Due to this, we are often not able to start labs until later or just hope our tests work even though we cannot run them.

Conclusion

What have you learned from this experience?

From this experience, we gained hands-on experience with a variety of testing techniques, which deepened our understanding of software verification. By combining boundary testing, MCDC, and exception handling, we ensured our system behaved predictably under both normal and abnormal inputs.

What improvements can be made in this experience in the future?
In the future, it would be helpful to include more complex triangle scenarios, such as those read form JSON or XML formats, and to automate test data generation. Adding a code coverage tool or visualization plugin could also further enhance test quality and understanding.