Names: Madison Betz and Kaiden Pollesch

Date: 2/18/2025

Assignment Title: Lab 3 Reachability, Infection, Propagation, and Revealability

Repository Link: https://github.com/msoe-SWE2721/swe2721-lab3-2025-121_betz_pollesch_lab3_2025.git

Introduction

In this lab we are creating code with a simple bug and designing tests along with the RIPR (Reachability, Infection, Propagation, and Revealability). We are making code that converts from numbers written out in text to digits and vice versa. The text numbers are to be in Spanish and tested with TestNG that we have also created. Each of the different conversions are to have one bug that is to be found with the RIPR in four tests with only the last test failing, therefore revealing the bug. One partner is implementing one of the conversion methods with the opposite test and the other partner is doing the other coding and testing.

Bug Identifications

Location/Class:Line	**Bug**
NumericStringConverterPartA():49	Incorrect logic to convert from a decimal to "punto" and instead remains as a decimal point in output
NumericStringConverterPartB(): 12-61-it will happen with all numbers	Incorrect logic to handle two or more of the same words in a sentence.

Testing Strategy

TestPartA: I was given the bug of incorrect logic to convert the character of a decimal point into the word "punto". To achieve each stage of the RIPR, my first test focused on reachability. I sent in a string with no numbers so that the program would run but no conversion would actually take place. For the infection stage, I went through the switch statement by having a number within a string that needed to be converted and having no decimals or periods within the statement. The test for the propagation stage has a period at the end which should not convert but be flagged as maybe needing conversion, so the test passes as the dot was not converted. The revelation test fails as expected as a decimal is sent into the program, but it is not converted correctly.

TestPartB: The code I was given had a bug when converting multiple digits in string form of the same number to the actual digit, causing all digits to be converted but only counted as one. To achieve each stage of the RIPR model, I first tested Reachability by providing a string with no numbers to ensure the function executed without triggering the bug. For Infection, I used a single-digit number to confirm the conversion logic was applied. In Propagation, I tested a multi-digit number, where the incorrect count occurred, proving the bug altered the program state. Finally, in Revelation, I compared the expected output to the actual result, confirming that the bug was observable due to the incorrect digit count.

Each person should explain how, given the bug, they went about achieving each stage of the RIPR model when testing. If for some reason one of the stages of RIPR cannot be implemented as a test case, clearly explain why this was the case.

Discussion

How did data providers help in writing these test cases?

Using data providers helped in writing these test cases as it makes clear which strings were input and what the expected output would be. It also allowed the tests to be more organized and the layout of them clearer to understand.

Was this task easier or harder than writing tests against a given specification?
This task was easier than writing tests against a given specification. We already knew what the bug was, so finding it with a test was much simpler. However, it was somewhat more difficult to create test cases that we knew would come upon the bug without actually failing a test or triggering the program to fail.

What type of test are you writing here? Why do you provide this answer?
The type of test we are writing here are unit tests. This is because we are testing an individual component of what would be a larger system. Each class is individually tested for expected behaviors. This is not testing the full system but only the partial functionality of a class and the code within it.

Things Gone Right / Things Gone Wrong

Things that went right with this lab was initially writing the code. We were able to implement the classes based on the instruction given to us and properly write tests that reached each stage of RIPR. This allowed us to add a bug in each that could be found by failing in the final test for revelation.
Things that went wrong in this lab were again setting up and downloading the code. This took several hours to fix where the code would not build or compile.

Conclusion

What have you learned from this experience?
From this experience we have learned more about TestNG and the RIPR testing structure. We wrote our tests in the AAA (Arrange, Act, and Assert) method with data providers to make the tests clear to understand and simple to execute and write. The RIPR stages of testing and finding a bug made it so we were able to see how bugs can make it past different types of tests and what tests will find bugs and fail.

What improvements can be made in this experience in the future?
Improvements that could be made to this experience in the future is using some other software than Gradle. Gradle causes a lot of issues when downloading code, so most of the time it will take hours to get it to build or compile correctly.