Test-Driven Development of the Calculator Class Report

Muganga Charles

February 2024

Approach

Red Phase:

- I began by designing unit tests for a non-existent Calculator class.
- I then focused on covering the core arithmetic operations (addition, subtraction, multiplication, division) and handling potential error cases (division by zero).

Green Phase:

- Here I implemented the Calculator class with either instance methods or static methods.
- I ensured all unit tests passed successfully.
- Then I prioritized code clarity and functionality in the initial implementation.

Refactor Phase:

- I reviewed the code for readability and optimization opportunities.
- I implemented the following refactorings:
  - Added docstrings for the class and methods to improve documentation
  - Enhanced variable naming (num1, num2) to increase clarity.
  - Adjusted spacing for consistency and adherence to PEP 8 guidelines.

Challenges

- **Choosing Between Method Types:** Deciding between instance methods and static methods required an understanding of the implications of statefulness in object-oriented programming.

Refactoring Rationale

- **Improved Documentation:** Docstrings provide valuable information about the usage and behavior of the class and its methods, promoting maintainability.
- **Enhanced Readability:** Descriptive variable names and consistent formatting contribute to code that is easier to understand, making collaboration and future modifications less complex.

Conclusion

Test-Driven Development (TDD) has proved to be an effective methodology, ensuring that the Calculator class met the specified requirements and promoting a focus on maintainability right from the start.