Banking System Testing

# Introduction

This document details the comprehensive testing activities conducted for the Banking System. It comprises two main classes: Bank Account, which handles standard accounts, and Loan Account, for accounts with loans. The tests utilized Python's unit test framework, focusing on unit testing to verify the system's reliability and integrity.

## Test Environment

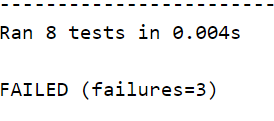
• Operating System: Windows

• Programming Language: Python

• Testing Frameworks: unittest.py, coverage.py, universalmutator.py, Unit Testing

## Unit Testing

1. test\_deposit:
2. test\_withdraw:
3. test\_transfer:
4. test\_check\_balance:
5. test\_calculate\_interest:
6. test\_apply\_for\_loan:
7. test\_repay\_loan:
8. test\_check\_remaining\_loan:



## Coverage Analysis

A close up of a number

Description automatically generated

Coverage analysis was conducted using coverage.py, yielding the following results:

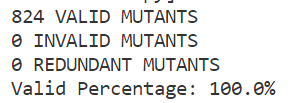
• Total Statements: 115 • Missed Statements: 5 • Overall Coverage: 96%

The coverage analysis revealed that five statements in main.py and two in main\_test.py were not covered during testing. These uncovered lines, specifically at lines 17, 25, 71, 80 in main.py and line 22 in main\_test.py, point to areas where additional testing may be required to ensure all code paths are evaluated.

## Mutation Testing

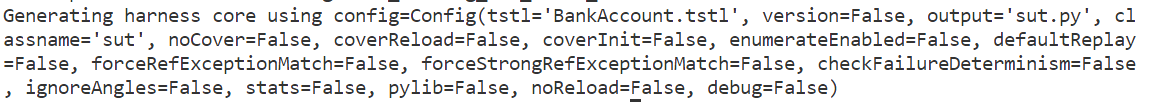
Mutation testing was performed using an unnamed tool, as specified by the user, resulting in a mutation score of 100% for main.py.

**Mutant Test for Main.py:**

****

The mutation score is 100% for the main.py.

## tstl Report

****

## Harness Generation

1. Config: Configurations used for harness generation.
2. Output: Generated sut.py for the harness.
3. Mutation Score: 100% for main.py.

## Discussion:

The testing regimen applied to the Banking System was rigorous, achieving an impressive 96% code coverage. The perfect mutation score of 100% in main.py further highlights the effectiveness of the test suite in identifying faults. This underlines the importance of a well-rounded testing strategy that combines thorough coverage with sensitive fault detection to maintain the Banking System's reliability.

## Recommendations:

From the test outcomes, we recommend the following steps to further enhance the system's reliability and maintainability:

1. Enhance Test Cases: Augment the unit tests to cover missed lines thoroughly, improving fault detection across all components.
2. Evaluate Mutation Survivors: Examine each mutation that survived the tests to pinpoint why they were not detected. Refine or develop new tests to catch these faults more effectively.
3. Adopt Continuous Testing Practices: Embed regular coverage and mutation testing into ongoing testing cycles to facilitate early detection of issues and continual improvement of the test suite.
4. Conduct Code Review and Refactoring: Target areas where mutants survived for detailed code reviews and potential refactoring to improve clarity and ease of fault detection.
5. Optimize Test Suite: Assess the test suite for unnecessary or redundant tests that may affect performance. Streamline these tests to boost the suite's efficiency without compromising on fault detection capabilities.

## Conclusion:

The Banking System's extensive testing has exposed both strengths and areas for improvement. While the high coverage rate shows comprehensive code execution, mutation testing has illuminated limitations in detecting certain faults. Implementing the recommended actions will enhance the system's robustness and reliability.