Software Testing November 27, 2023

Mutation Testing on Source code of Calculator

Instructor: Prof. Meenakshi D Souza Term I 2023-'24: Project Work

Team members

- IMT2020110 Vivek Tangudu
- IMT2020124 Akhil Tavva

1 Project Aim

The goal of the project is to utilize open-source tools to perform Mutation Testing on a real-world software project.

2 Introduction

Mutation testing is a type of testing in which the source code is slightly altered (each change is referred to as a mutant). The purpose of mutation testing is to "kill" these mutants, i.e., demonstrate how even minor alterations to the code can affect how the program runs and, consequently, the output that it generates. Two methods exist for "killing" a mutant:

1. Kill Weak Mutant: In this case, the memory state of the program after executing the mutated statement differs from the memory state of the program before executing the mutated statement. It's worth noting that in this case, the program's output on a test case can remain unchanged regardless of whether a program statement was mutated or not.

2. Kill a Strongly Mutant: Here, the output of the program on a test case, when a statement was mutated and not mutated, must change. Notice that when we kill a mutant strongly, the error propagates through the program and we notice this by seeing different outputs in the presence and absence of the mutant.

Here, in this project we aim to kill the mutants strongly.

3 Source Code Description

Github repo link: https://github.com/vivektangudu123/Software_testing_project

3.1 Calculator.java

We used a source code from the Calculator that has some basic functionalities in a calculator. We have modified the code such that it can handle errors and throw errors for invalid inputs.

- **add(double, double)**: Takes two double's as input argument and returns their algorithmic sum.
- **mul(double, double)**: Takes two double's as input argument and returns their multiplied sum. (Similarly, functions for subtracting, dividing, modulus, raise-to-x, root-to-x).
- **squareroot(double)**: Takes an integer as input and returns its squareroot of that number.
- **sin(double)**: Takes an integer as input and returns the sin value of input.

- **asin(double)**: Takes an integer as input and returns the angle when sin inverse of input is taken. (Similarly, functions for square, cube, dividing, absolute, log, exponential, round, cosine, tangent, contangent, secant, cosecant, acosine, atangent).

For every function, we have a corresponding function that gives input to the mathematical functions. "=" is the end of a mathematical instruction. If there is no "=' it goes on to use the same mathematical function.

3.2 BankAccount.java

The BankAccount class represents a basic bank account with currency conversion capabilities.It uses the currencycoverter class it give the value of money.

It includes the following functionalities:

3.2.1 Attributes:

- **balance:** Represents the balance in the bank account.
- **converter**: An instance of the CurrencyConverter class for handling currency conversions.
- **defaultCurrency:** Represents the default currency of the bank account.

3.2.2 Constructor:

- **public BankAccount(String defaultCurrency):** Initializes a new bank account with the specified default currency.

3.2.3 Methods:

- **public void addMoney(double amount, String currency):** Adds money to the account in the specified currency, converting it to the default currency.
- **public void setDefaultCurrency(String currency):** Changes the default currency of the account, converting the existing balance to the new default currency.
- **public void convertDefaultCurrencyTo(String currency):** Converts the default currency of the account to the specified currency.
- **public double getBalance()**: Returns the current balance in the default currency.

3.3 CurrencyConverter.java

The CurrencyConverter class provides functionality for currency conversion. It uses the class Calculator to do arthimetic functions.

Key features include:

3.3.1 Attributes:

- **exchangeRates:** A Map that stores exchange rates for different currencies.
- **defaultCurrency:** Represents the default currency for conversions.

3.3.2 Constructor

- **public CurrencyConverter():** Initializes a new currency converter with default exchange rates.

3.3.3 Methods:

- public void setExchangeRate(String currency, double rate): Sets the exchange rate for a specified currency.
- **public double convertToDefaultCurrency(double amount, String fromCurrency) throws CurrencyNotFoundException:** Converts an amount from a specified currency to the default currency and throws an exception if the currency is not found.
- **public double convertFromDefaultCurrency(double amount, String toCurrency) throws CurrencyNotFoundException:** Converts an amount from the default currency to a specified currency and throws an exception if the currency is not found.
- **public double convert(double amount, String fromCurrency, String toCurrency) throws CurrencyNotFoundException:** Converts an amount from one currency to another and throws an exception if the currency is not found.

3.4 OutOfDomainException.java

This class allows developers to create instances of the OutOfDomainException class, which can be thrown to indicate exceptional situations related to being out of a specific domain or range, typically during program execution. The exception can be caught and handled elsewhere in the code to address such exceptional cases.

3.5 MyRuntimeException.java

This exception can be thrown in situations where a runtime error occurs and needs to be handled explicitly by the code that catches this specific exception. It allows for more tailored exception handling and communication of runtime issues through the provided message.

3.6 InvalidInputException.java

This exception can be thrown to indicate exceptional situations specifically related to invalid input conditions during program execution. When this exception is thrown, it can be caught and handled elsewhere in the code to manage issues arising from invalid inputs.

3.7 CurrencyNotFoundException.java

This class is intended to be thrown in situations where a currency-related issue occurs and needs to be handled explicitly by the code that catches this specific exception. Being a checked exception, it prompts the developers to handle this exception in their code, improving error handling and robustness.

4 Active mutators

PIT, which by default provides a set of mutation operators. The operators listed for all the Java files are:

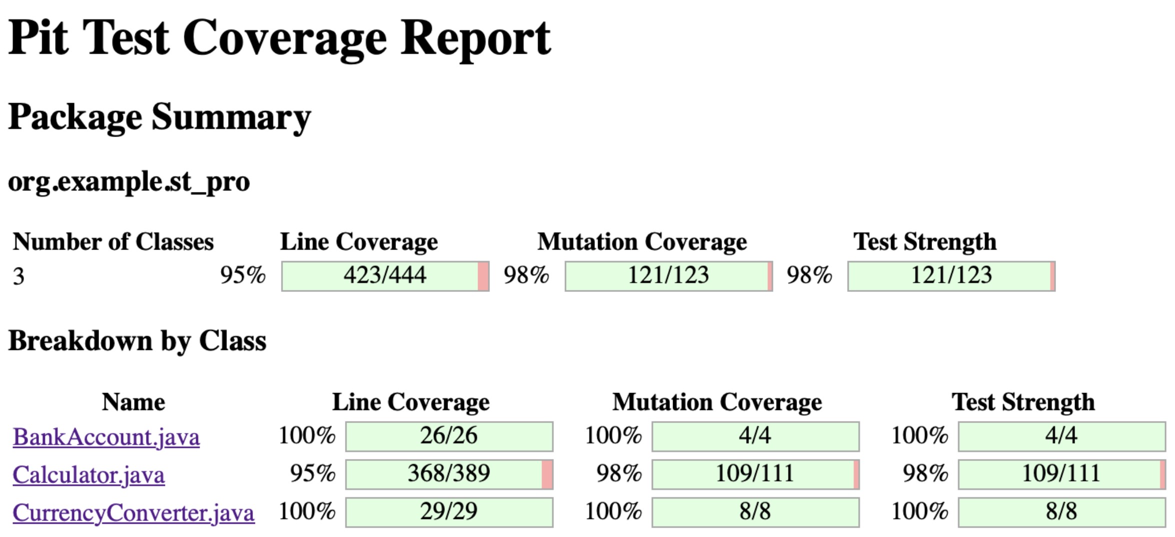
- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS

- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

5 PITest Results

Class Name	Mutants Killed	% Mutants Killed	Lines of Code in Class (approx).
BankAccount.java	4/4	100	51
Calculator.java	109/111	98	1029
CurrencyConverter.java	8/8	100	64

6 Testing Summaries



Pit Test Coverage Report

Package Summary

org.example.st_pro

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
3	95%	423/444	98%	121/123	98%	121/123

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
BankAccount.java	100%	26/26	100%	4/4	100%	4/4
Calculator.java	95%	368/389	98%	109/111	98%	109/111
CurrencyConverter.java	100%	29/29	100%	8/8	100%	8/8

Figure 1:

7 Execution

7.1 How to Execute mutation testing

Run the following commands:

1. mvn clean install
2. mvn test −compile org.pitest:pitest −maven : mutationCoverage

7.2 Contributions

Akhil:

- Worked on the final project report's curation.
- Worked on creating the test cases for Calculator java file and updating Calculator file to obtain more code coverage.

Vivek:

- Worked on Calculator, Bank Account and Currency Converter code and test files.
- Worked on the final project report's curation.

8 Refernces

1. Maven and JDK Setup:https://www.digitalocean.com/community/ tutorials/install-maven-linux-ubuntu
2. Junit Assert: https://junit.org/junit4/javadoc/4.13/org/junit/Assert.html
3. PITclipse: https://github.com/pitest/pitclipse
4. Source Code: Project Source code