Nafis Fuad

[Email address]

Test report

[Document subtitle]

LETTER OF TRANSMITTAL

Abdus Satter

Lecturer

Institute of Information Technology

University of Dhaka

**Subject: Submission of final report on Software Testing**

Dear Sir,

With due respect, we are pleased to submit the final report on Digital Software Testing. Although this report may have shortcomings I did try our level best to produce an acceptable report. I would be highly obliged if you overlooked our mistakes and accepted our effort.

Sincerely yours,

Mridha Md. Nafis Fuad

BSSE 0920

Institute of Information Technology

University of Dhaka

**Acknowledgement**

I am highly indebted for getting such a tremendous opportunity to prepare the report on Software Testing. I would like to thank whole-heartedly our course teacher, Mr. Abdus Satter, Lecturer, Institute of Information Technology, University Of Dhaka, for giving us guidelines about how I can prepare this report.

**Abstract**

This paper is an introduction to Design Patterns. Patterns are recent software engineering problem solving discipline that emerged from object oriented community. The primary purpose of pattern is communicating design insights and making patterns coherent and easier to understand.

Software quality is considered to be one of the most important concerns of software production teams. Additionally, design patterns are documented solutions to common design problems that are expected to enhance software quality. Until now, the results on the effect of design patterns on software quality are controversial.

This study aims to propose a methodology for comparing design patterns to alternative designs with an analytical method. Additionally, the study illustrates the methodology by comparing three design patterns with two alternative solutions, with respect to several quality attributes.

Software quality is considered to be one of the most important concerns of software production teams. Additionally, design patterns are documented solutions to common design problems that are expected to enhance software quality. Until now, the results on the effect of design patterns on software quality are controversial.

This study aims to propose a methodology for comparing design patterns to alternative designs with an analytical method. Additionally, the study illustrates the methodology by comparing three design patterns with two alternative solutions, with respect to several quality attributes.

The paper introduces a theoretical/analytical methodology to compare sets of "canonical" solutions to design problems. The study is theoretical in the sense that the solutions are disconnected from real systems, even though they stem from concrete problems. The study is analytical in the sense that the solutions are compared based on their possible numbers of classes and on equations representing the values of the various structural quality attributes in function of these numbers of classes. The exploratory designs have been produced by studying the literature, by investigating open-source projects and by using design patterns. In addition to that, we have created a tool that helps practitioners in choosing the optimal design solution, according to their special needs.

The results of our research suggest that the decision of applying a design pattern is usually a trade-off, because patterns are not universally good or bad. Patterns typically improve certain aspects of software quality, while they might weaken some other.

Concluding the proposed methodology is applicable for comparing patterns and alternative designs, and highlights existing threshold that when surpassed the design pattern is getting more or less beneficial than the alternative design. More specifically, the identification of such thresholds can become very useful for decision making during system design and refactoring

# Chapter 1: Test Planning

## About the project:

Java Algebra System (JAS) is a multivariate computer algebra system built using Java. JAS handles the following scenarios in both fraction and decimal mode:

* Commutative arithmetic & algebra
* Nested unary operation simplification
* Algebraic domain validation
* Trigonometric simplification
* Logarithmic exponential simplification
* Basic irrational/rational number arithmetic
* First Derivative
* Logarithmic differentiation
* Implicit differentiation
* nth derivative

## Number of test cases and their cost:

As a simple project is has very simple methods. So to test the project I have created 150 (not intentionally) test cases. The methods sometimes are modified to make them public or test an object inside a method. Such methods were a bit trickier to test.

## Areas of risk:

1. **Defect rate**: The expected defect rate has been increased due to lack of available requirement specification document and thus lack of proper test case design.
2. **Number of Test Cases**: Due to lack of skills and experience I probably could not generate the sufficient number of test cases.

## Test completion:

If the project has less than 10% failure then the project is successful. Due to lack of experience I tested the project with unit testing. There’s more sufficient ways for testing upon which the completion depends.

# Chapter 2: Test Design

## Test Objectives:

The main objective of testing the project is to check either users can run various arithmetic and algebraic operations on the system or not. So, the methods for searching, parsing, extracting nodes, mathematical operations are in high priority for testing. Objectives of this testing is to test:

* Whether the parsing is done properly
* Whether the system can handle wrong/invalid input
* Robustness of the system by providing erroneous parameters
* Whether each type of parsing e.g. Parenthesis, operator etc. is done properly
* Detection of appropriate operators and applying them to operands
* Testing results of mathematical logic

## Items to be tested:

* Creating valid string for algebraic operation
* Parsing string
* Formatting operations
* Indexing of resources
* Robustness in case of invalid input

## Design Techniques:

The technique for testing this project is Black box testing. I have used equivalence class testing and robust testing for the project.

## Creating test cases and Test data:

1. Test cases to test parse the input statement. So I have to test parsing input statement, removing parenthesis and updating tags in operators
2. Test cases to test searching for an existing constants and operations. So, test cases should be created to test the validity of the mentioned constants and operations
3. Test cases to test sorting the contacts. So test cases are needed to sort the elements in ascending order and descending order
4. Test cases to test filtering search result. To test this test cases should be created to test if the filtering is done perfectly using different parameters
5. Test cases to test adding new functions any existing contact. Test cases should be created to check if a function exists or not, after removal if the updating is done perfectly or not
6. Test cases to test program crashes from invalid input. Test cases should be created to check if proper error handling is done in unit functions and the program does not crash unexpectedly

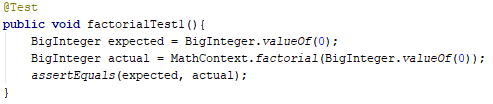
## Tools used to generate test cases:

To do unit testing **Junit4** has been used as a tool. JUnit is a unit testing framework for the Java programming language. As the project is developed in Java, so JUnit testing has been useful to test this project.

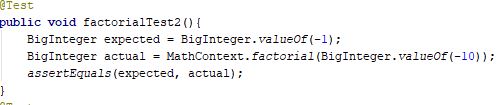
# Chapter 3: Test Planning

## Class MathContext:

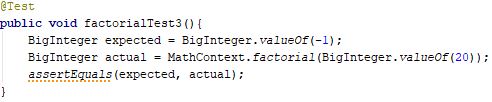
### Test Case: T001



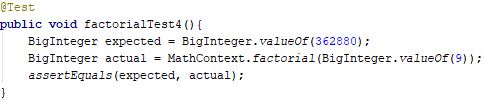
### Test Case: T002



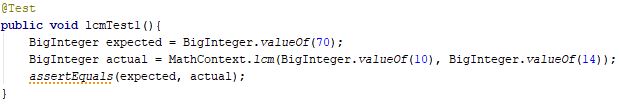
### Test Case: T003



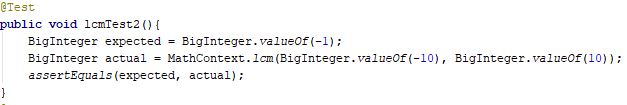
### Test Case: T004



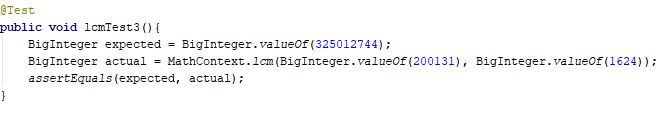
### Test Case: T005



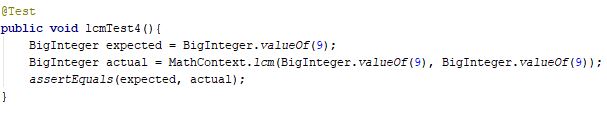
### Test Case: T006



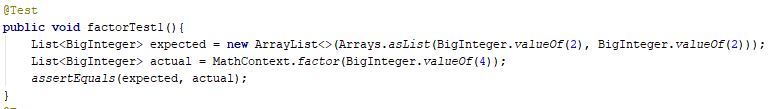
### Test Case: T007



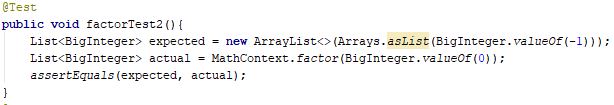
### Test Case: T008



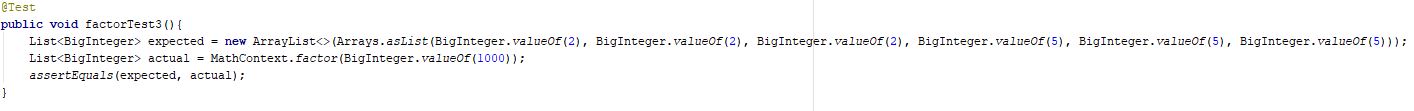
### Test Case: T009



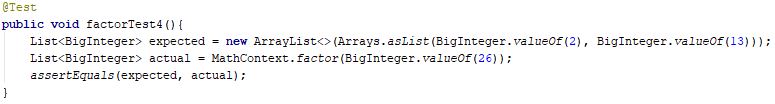
### Test Case: T010



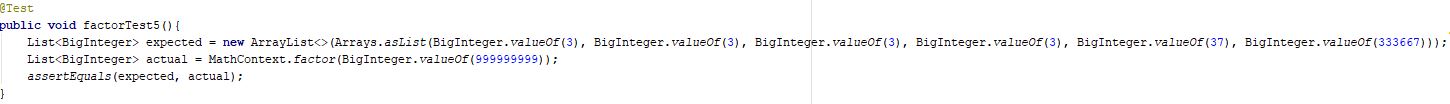
### Test Case: T011



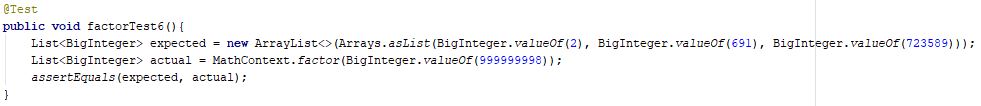
### Test Case: T012



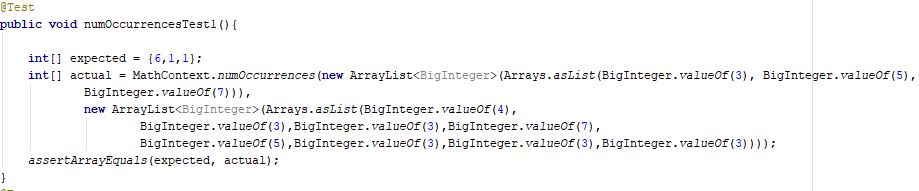
### Test Case: T013



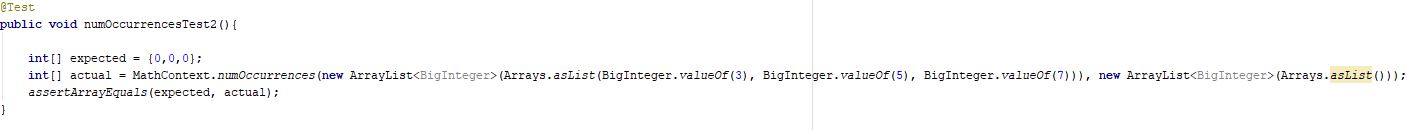
### Test Case: T014



### Test Case: T015

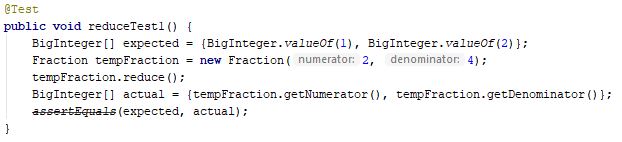


### Test Case: T016

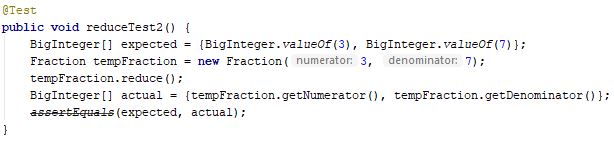


## Class Fraction:

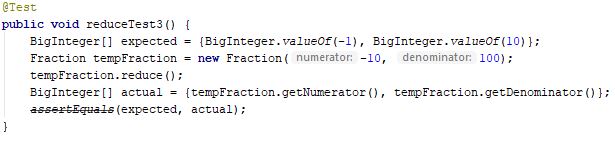
### Test Case: T016



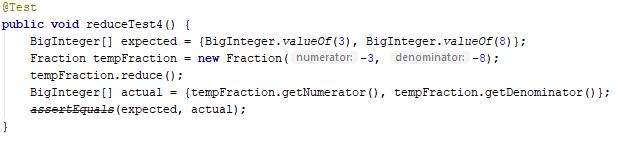
### Test Case: T017



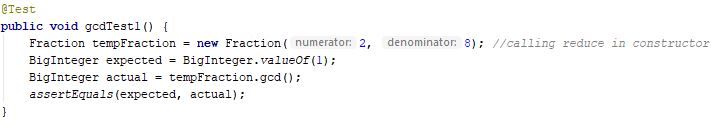
### Test Case: T018



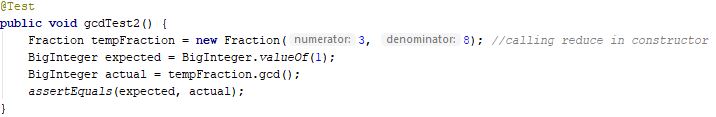
### Test Case: T019



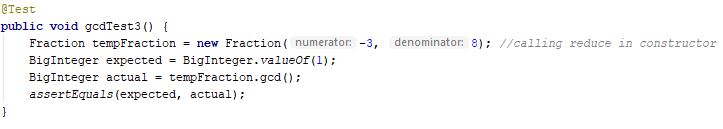
### Test Case: T020



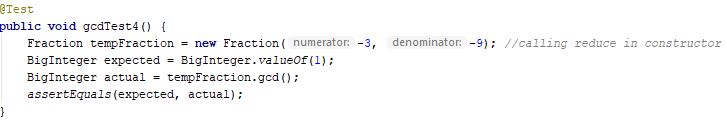
### Test Case: T021



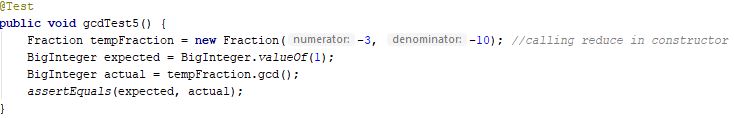
### Test Case: T022



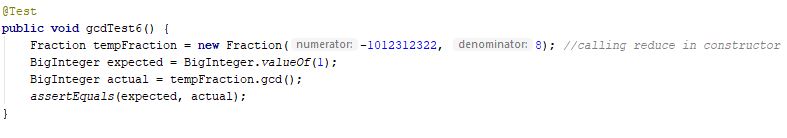
### Test Case: T023



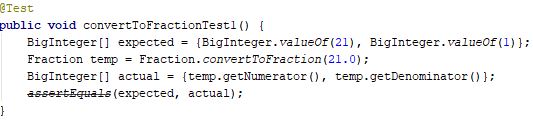
### Test Case: T024



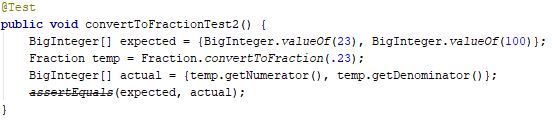
### Test Case: T025



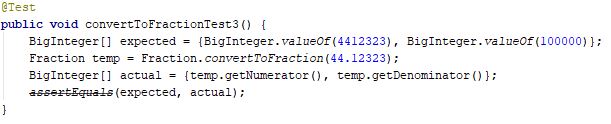
### Test Case: T026



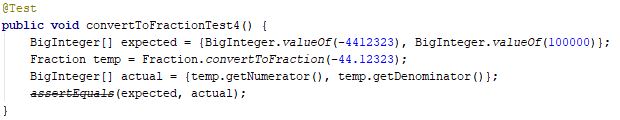
### Test Case: T027



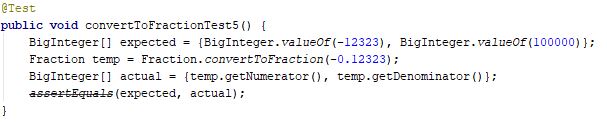
### Test Case: T028



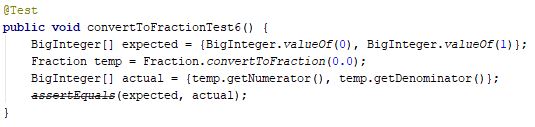
### Test Case: T029



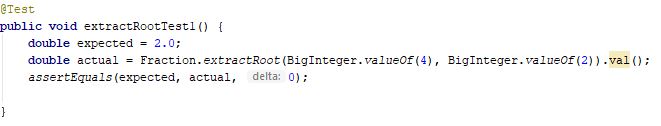
### Test Case: T030



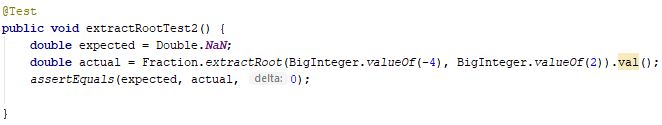
### Test Case: T031



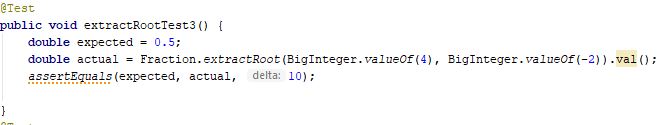
### Test Case: T032



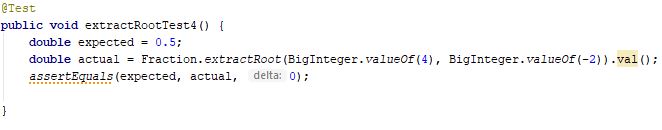
### Test Case: T033



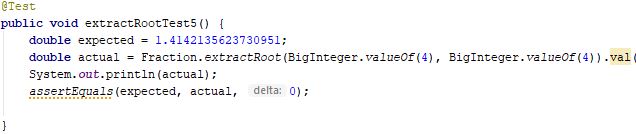
### Test Case: T034



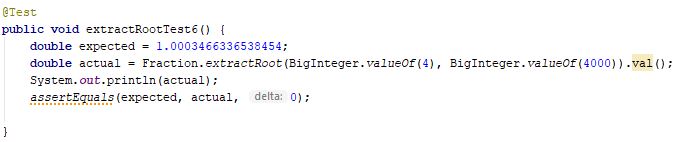
### Test Case: T035



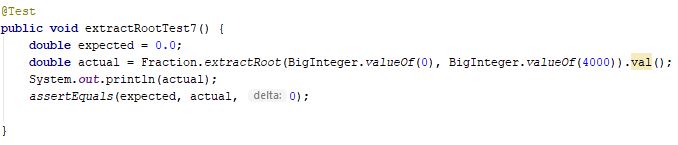
### Test Case: T036



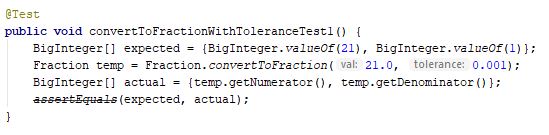
### Test Case: T037



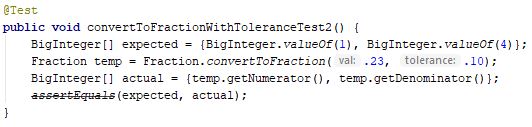
### Test Case: T038



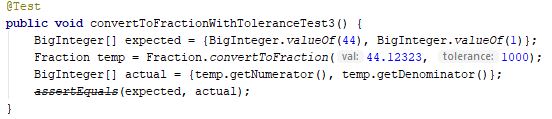
### Test Case: T039



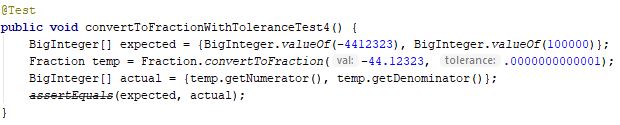
### Test Case: T040



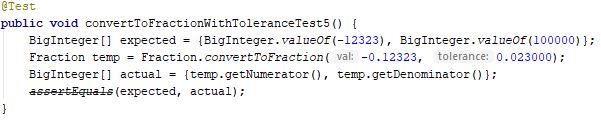
### Test Case: T041



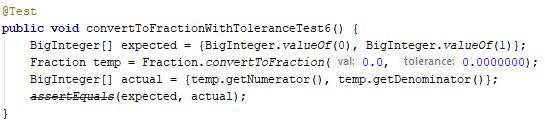
### Test Case: T042



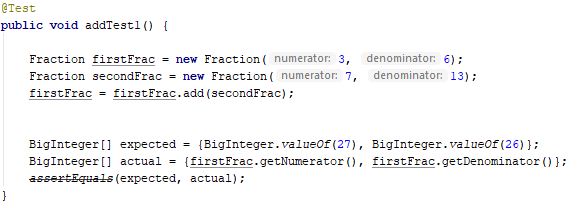
### Test Case: T043



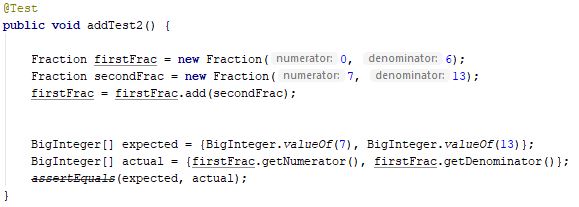
### Test Case: T044



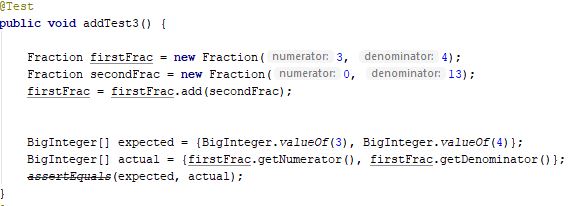
### Test Case: T045



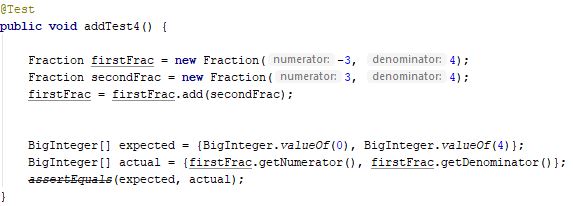
### Test Case: T046



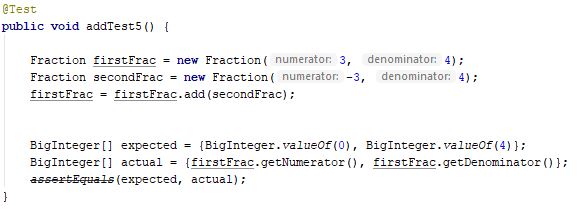
### Test Case: T047



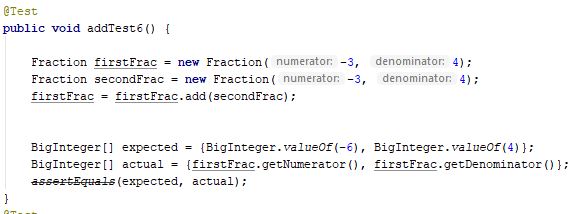
### Test Case: T048



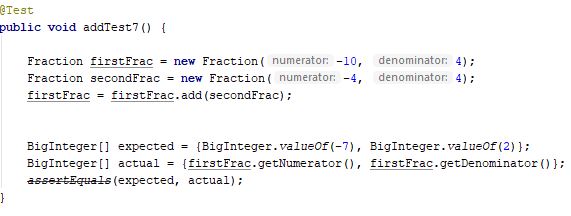
### Test Case: T049



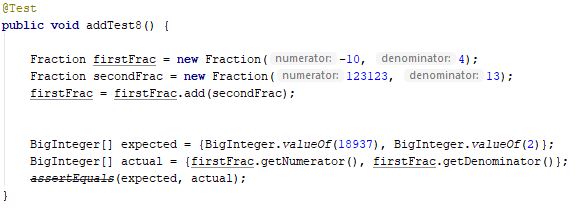
### Test Case: T050



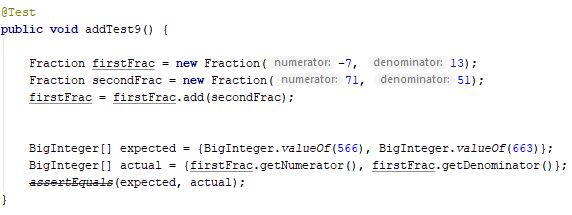
### Test Case: T051



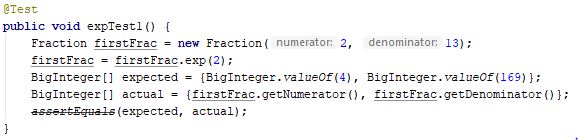
### Test Case: T052



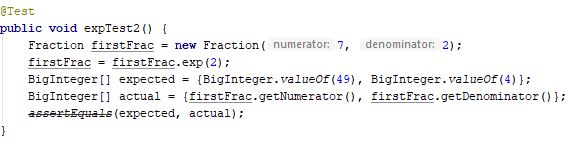
### Test Case: T053



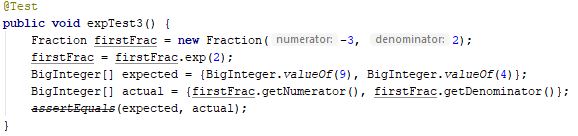
### Test Case: T054



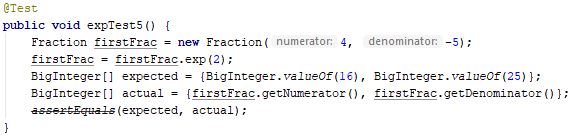
### Test Case: T055



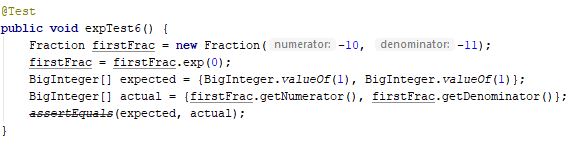
### Test Case: T056



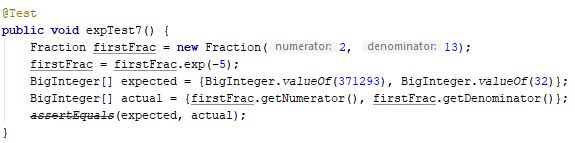
### Test Case: T057



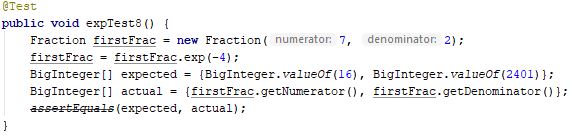
### Test Case: T058



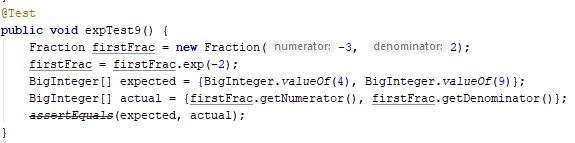
### Test Case: T059



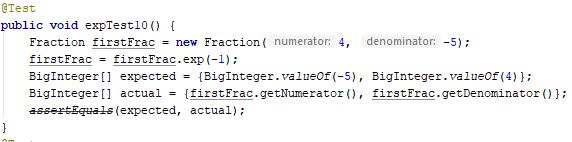
### Test Case: T060



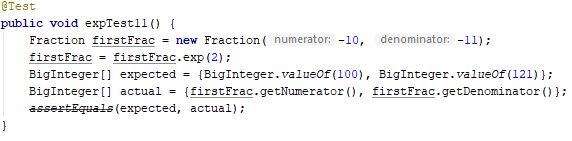
### Test Case: T061



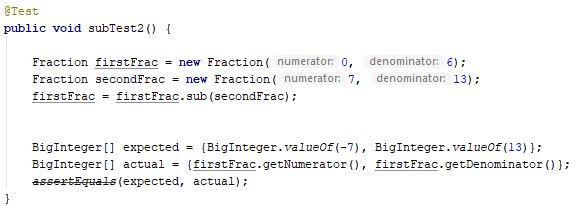
### Test Case: T062



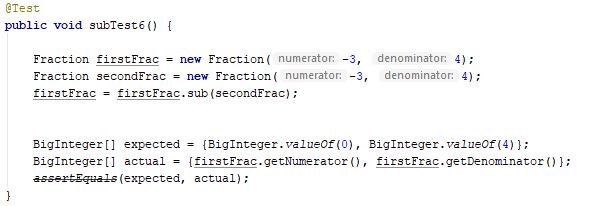
### Test Case: T063



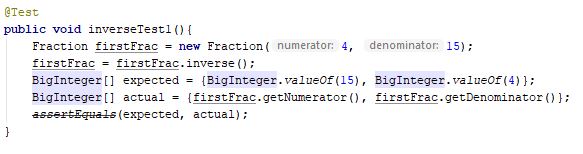
### Test Case: T064



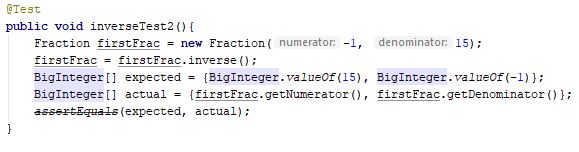
### Test Case: T065



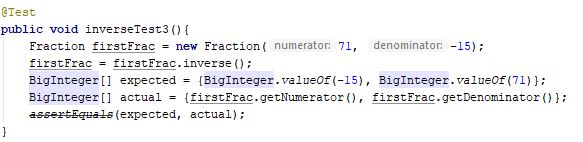
### Test Case: T066



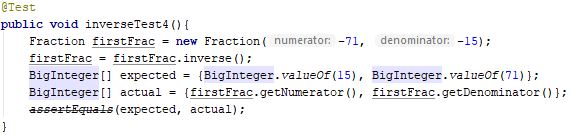
### Test Case: T067



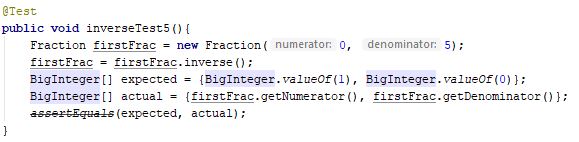
### Test Case: T068



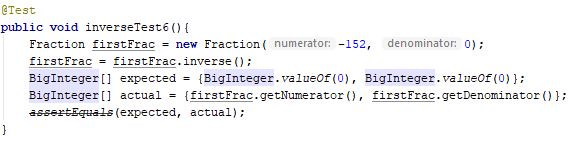
### Test Case: T069



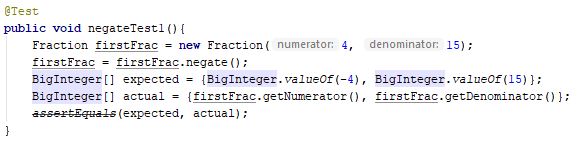
### Test Case: T070



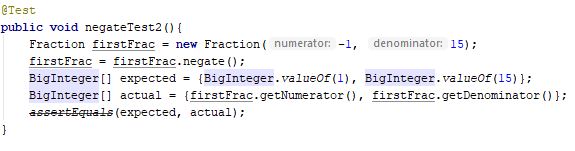
### Test Case: T071



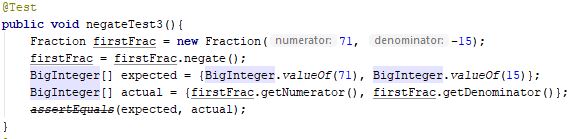
### Test Case: T072



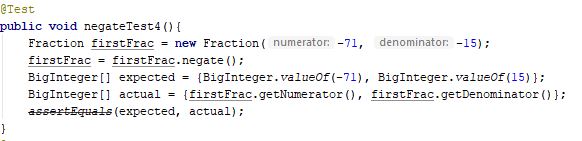
### Test Case: T073



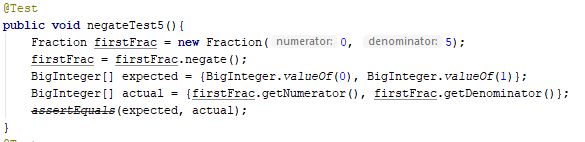
### Test Case: T074



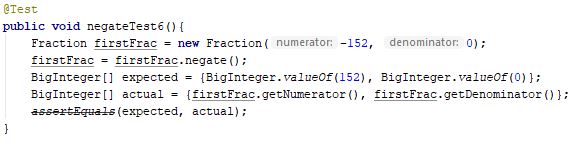
### Test Case: T075



### Test Case: T076

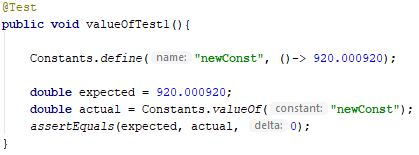


### Test Case: T077



## Class Constants:

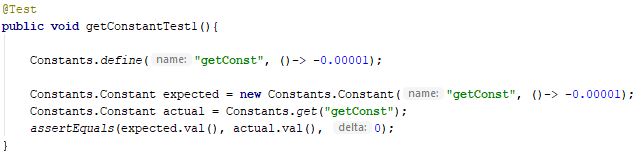
### Test Case: T078



### Test Case: T079

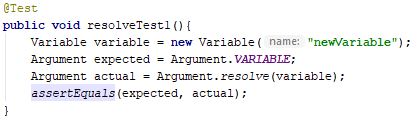


### Test Case: T080

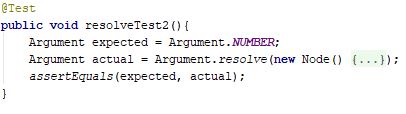


## Class Argument:

### Test Case: T081

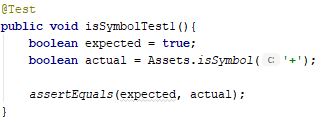


### Test Case: T082

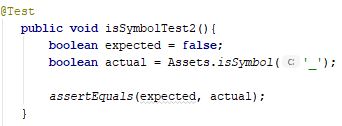


## Class Assets:

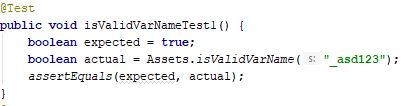
### Test Case: T083



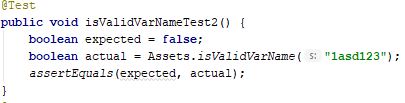
### Test Case: T084



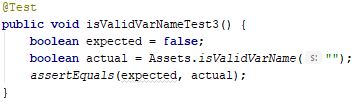
### Test Case: T085



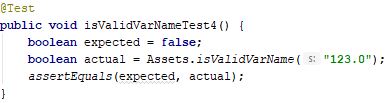
### Test Case: T086



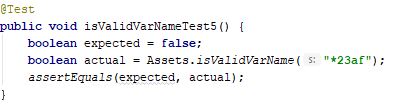
### Test Case: T087



### Test Case: T088

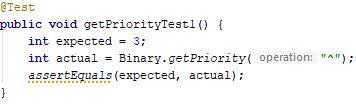


### Test Case: T089

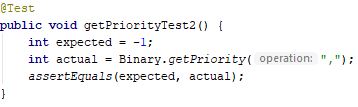


## Class Binary:

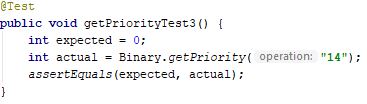
### Test Case: T090



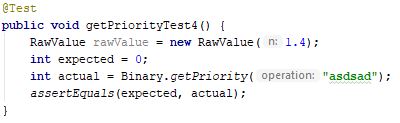
### Test Case: T091



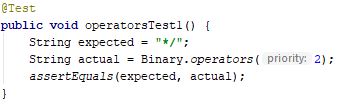
### Test Case: T092



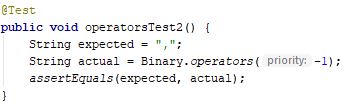
### Test Case: T083



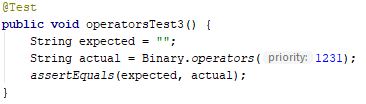
### Test Case: T083



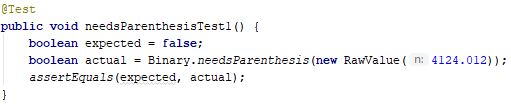
### Test Case: T083



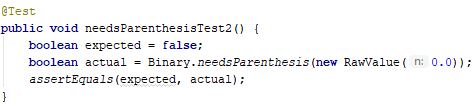
### Test Case: T083



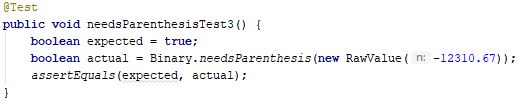
### Test Case: T083



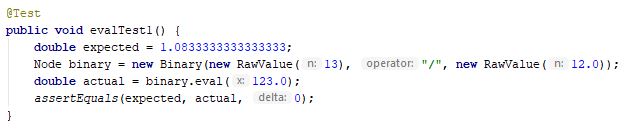
### Test Case: T083



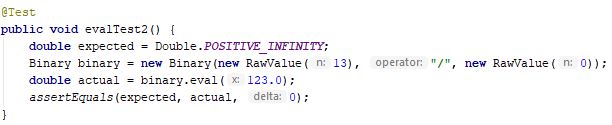
### Test Case: T083



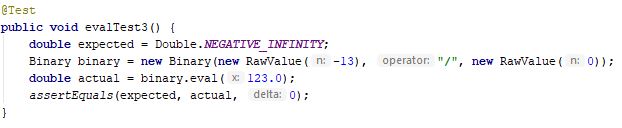
### Test Case: T083



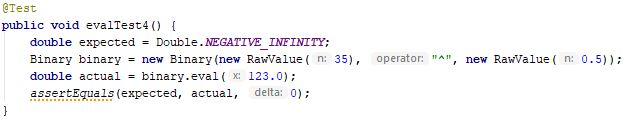
### Test Case: T083



### Test Case: T083



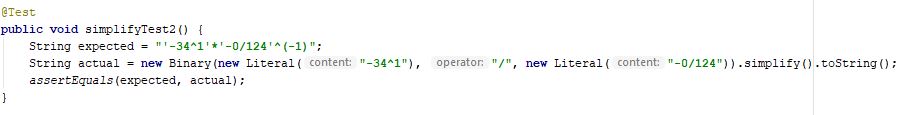
### Test Case: T083



### Test Case: T083



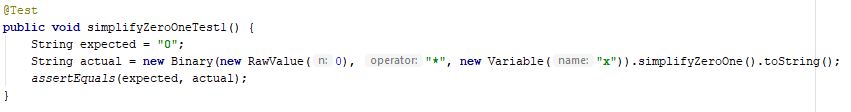
### Test Case: T083



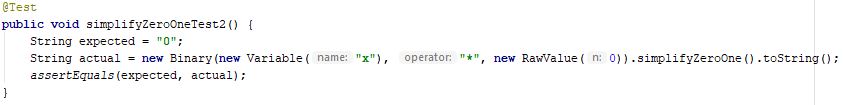
### Test Case: T083



### Test Case: T083



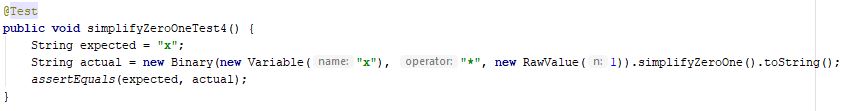
### Test Case: T083



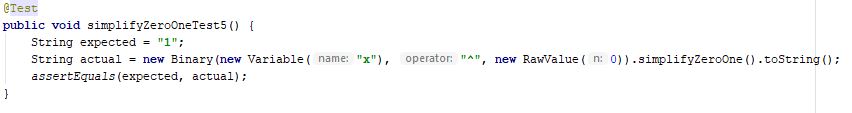
### Test Case: T083



### Test Case: T083



### Test Case: T083



### Test Case: T083

