

**Department of Electrical & Computer Engineering**

**North South University**

**Project Report**

**Project Name:** Calculator App

**Submitted by: Team - 03**

|  |  |  |
| --- | --- | --- |
| **Team Details** | | |
| **1** | Kazi Sakib Ahmad | ID: 1510702042 |
| **2** | Sohanur Rahman | ID: 1510464642 |

**GitHub Link:** <https://github.com/nsuspring2019cse427/Group03>

**Course:** CSE427 (Software Quality Assurance & Testing)

**Section:** 01

**Semester:** Spring 2019

**Submitted to:** Shaikh Shawon Arefin Shimon (SAS3)

Lecturer, Department of ECE

# **Project Description**

## **Introduction**

The project is mainly emphasized on implementing the learning outcomes of the academic course **‘Software Quality Assurance & Testing (CSE427)’.** In order to meet project checkpoints we selected a simple Calculator App, which was developed in Android platform and further tested it following software testing methodologies. Initially we have implemented the basic and obvious features of a Calculator. Our primary focus was on unit testing the basic features and methods of the app.

## **Background and Product Context**

This calculator app we choose to work with was developed about 5-6 months ago as a test project. This app is based on Android platform and has the basic functionalities (i.e. Addition, Subtraction, Multiplication, and Division) of a calculator. As an Android app we choose to implement its full backend using Java. It has a user interface which seems very simple to its users.

## **Testing Aspects We Implemented**

We have successfully implemented following tasks as testing aspects:

* Unit testing each JAVA methods implemented in the existing project using JUnit.
* Catching the uncaught exception.
* Integration testing.
* Functionality testing.
* Input space partitioning.
* Graph partitioning.
* Fixing out the existing bugs after unit testing.

## **Tools/Frameworks Used:**

* JUnit 4
* Android Studio (IDE)
* Eclipse (IDE)

# **Input Space Partitioning**

**Inputs Characteristics:**

1) Integers

2) Float

3) Double

There are two approaches to do input space portioning. One is interface-based input domain modeling and another one is functionality-based input domain modeling. We applied interface-based input domain modeling in our project. We first identified our input characteristics & limited each characteristic into single parameters in our code. We listed the values from each characteristic block & used those values to run input space partition test codes in our project. Thus, we wrote 30 test cases & test codes in order to do input space partitioning in the “CalculatorTest.java” file of the project. The tables contain values of each characteristic blocks are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 1: Partition of Inputs for all Addition, Subtraction, Multiplication, Division** | | | |
| **Partition** | **Positive** | **ZERO** | **Negative** |
| a) Inputs are integers | 6, 7, 18, 302, 684,  1248, 39852 | 0 | -6, -18, - 302  -4731, -98432 |
| b) Inputs are floats | 6.2f, 57.6f, 872.61f,  1123.5f, 12341.23f | 0 | -6.2f, -57.6f, -872.61f  -7136.87f, -72341.21f |
| c) Inputs are doubles | 2459.432,64936.0003,254.3,54.3  , 500.365697, 10.2556 | 0 | -2459.432, -67395.4323, -6311.501, -321.245 |

|  |  |
| --- | --- |
| **Table 2: Partition for Addition Method** | |
| **Characteristics** | **Blocks of Values** |
| i) inputs with integer & double | 1000, 100.26 |
| ii) inputs with positive & negative values | 32565, -625 |

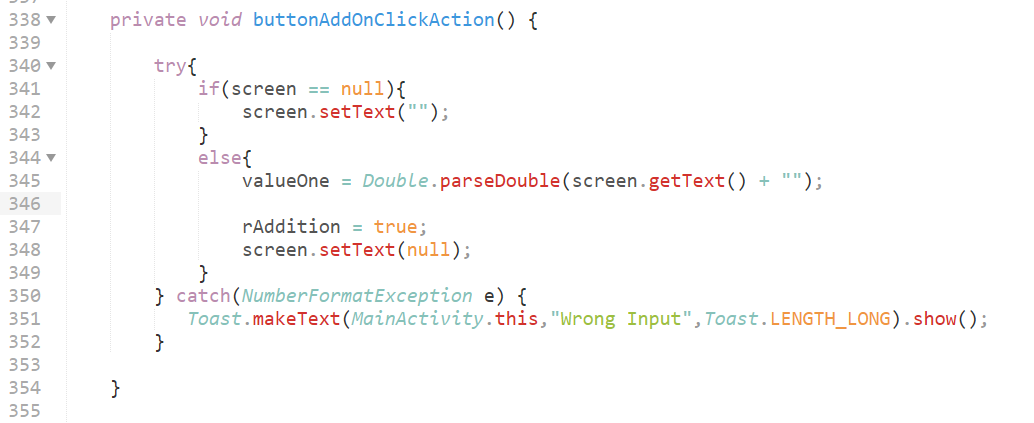
|  |  |
| --- | --- |
| **Table 3: Partition for Subtraction Method** | |
| **Characteristics** | **Blocks of Values** |
| i) inputs with integer & double | 10000.648, 200 |
| ii) inputs with positive & negative values | 32565, -625 |

|  |  |
| --- | --- |
| **Table 4: Partition for Multiplication Method** | |
| **Characteristics** | **Blocks of Values** |
| i) inputs with integer & double | 10000.648, 200 |
| ii) inputs with positive & negative values | -24, 649 |

|  |  |
| --- | --- |
| **Table 5: Partition for Division Method** | |
| **Characteristics** | **Blocks of Values** |
| i) inputs with numerator zero | 0.0, -321.245 |
| ii) inputs with positive & negative double values | 16911.365, -250.956 |

# **Graph Partitioning**

Graph partitioning of the piece of code under the method buttonAddOnClickAction() [line 338 – 354 of MainActivity.java ]



**Figure 1: Snapshot of buttonAddOnClickAction()**

**B**

**A**

**C**

**D**

**E**

Figure 2: Method buttonAddOnClickAction() with node defined

**E**

**D**

**C**

**B**

**!B**

**A**

**!A**

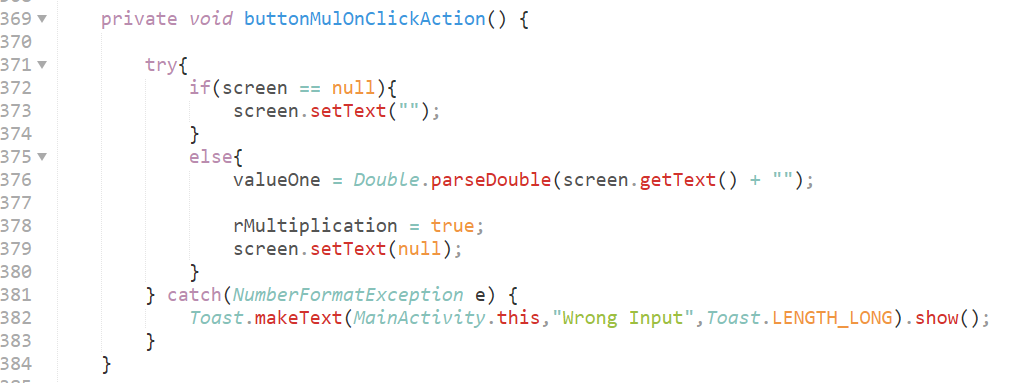
 Graph partitioning of the piece of code under the method buttonMulOnClickAction()

Figure 3: Method buttonMulOnClickAction() with node defined

**A**

**B**

**C**

**D**

**E**

**E**

**D**

**C**

**B**

**!B**

**A**

**!A**

Graph partitioning of the piece of code under the method buttonDivOnClickAction()

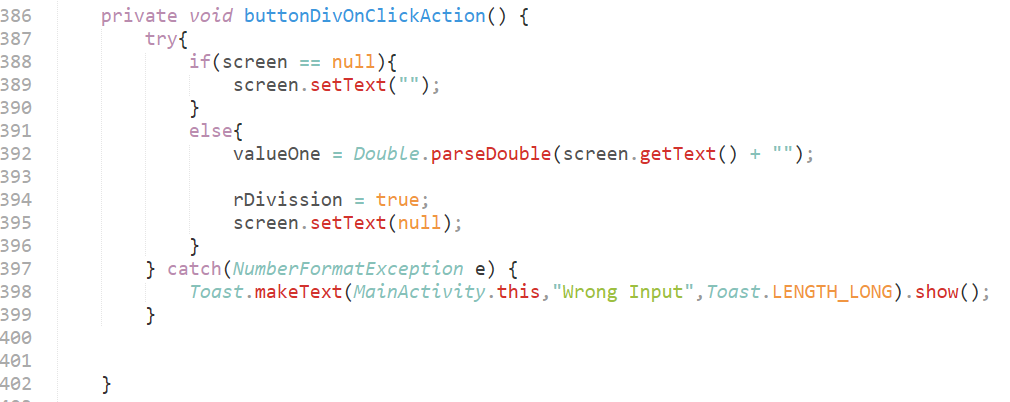


Figure 4: Method buttonDivOnClickAction() with node defined

**A**

**B**

**C**

**D**

**E**

**E**

**D**

**C**

**B**

**!B**

**A**

**!A**

**Node Coverage:**

|  |  |
| --- | --- |
| **Test Paths** | **Coverage** |
| t1 | [1,2,7] |
| t2 | [1,3,5,6,7] |
| t3 | [1,3,4,6,7] |

Test suit T = {t1, t2, t3}

**Edge Coverage:** Test Paths of Edge coverage are same as Node Coverage for this graph

**Edge-Pair Coverage:**

|  |  |
| --- | --- |
| **Test Paths** | **Test Requirements that are toured by test paths directly** |
| [1, 2, 7] | [1, 2, 7] |
| [1, 3, 4, 6, 7] | [1, 3, 4], [3, 4, 6], [4, 6, 7] |
| [1, 3, 5, 6, 7] | [1, 3, 5], [3, 5, 6], [5, 6, 7] |