CS5374 Software Verification and Validation
Fall 2013 Assignment 6 Report
Ka Son Chan

Purpose of the Assignment

This assignment focuses on formal verification. The assignment uses Java Modeling Language (JML) http://www.eecs.ucf.edu/~leavens/JML/examples.shtml to develop formal specification of a given class or method and then uses the pre-conditions and post-conditions to assert about the correctness of some properties. The formal specifications are written in JML language and are embedded into a given method as pre and post-conditions.

Contracts for Scala

According to Martin Odersky's Contracts for Scala, "Contracts are partial specifications that can be added to program code. They are checked by compilers or other tools, or, more commonly, by runtime checks. **Languages such as Eiffel, JML. pr Spec# support contracts natively. Scala does not. Instead, Scala provides flexible syntax that helps in writing high level libraries which can often mimic true language extensions.**"

Contracts as Code

The standard approach to writing specification like statements in Scala relies on four operations, all defined in Scala's standard `Predef` object, which is imported by default. They are:

- `assert(cond)` Throws an `AssertionError` if the given condition `cond` is false.
- `assume(cond)` Like assert, but is treated as an assumption (precondition) rather than asn assertion (postcondition) for program verifiers.
- `require(cond)` Throws an `IllegalArgumentException` if the given condition `cond` is false.
- `expr ensuring pred` Applies boolean-valued function `pred` to `expr`. If the result is `true`, the value of `expr` is returned; otherwise an `AssertionError` is thrown.

Testing Environment

All of the tests written in this report are executed and tested in Ubuntu 13.10 with Eclipse:

- No LSB modules are available.
- Distributor ID: Ubuntu
- Description: Ubuntu 13.10
- Release: 13.10
- Codename: saucy

Prerequisites and steps for functional tests

- You have installed Eclipse, the Android SDK, and ADT.
- You have been able to deploy a Java application to an Android device (physical or virtual).

Step 1: Install Scala-IDE

- You have installed the Scala-IDE that matches your version of Eclipse.
- Scala IDE Lithium works with Eclipse 4.2 and 4.3 (Juno and Kepler). http://download.scala-ide.org/nightly-scala-ide-4.0.x-210x

Step 2: Install AndroidProguardScala

- Point Eclipse to the update site at https://androidproguardscala.s3.amazonaws.com/UpdateSiteForAndroidProguardScala and install.

Step 3: Download the zip files MatrixCalculator and MatrixCalculatorTest

- Unzip both MatrixCalculator and MatrixCalculatorTest files.
- Import them into Eclipse environment.

Step 4: Check MatrixCalculator Manifest

- As shown in Figure U1 below, Debuggable should equals to true in MatrixCalculator AndroidManifest.xml file.

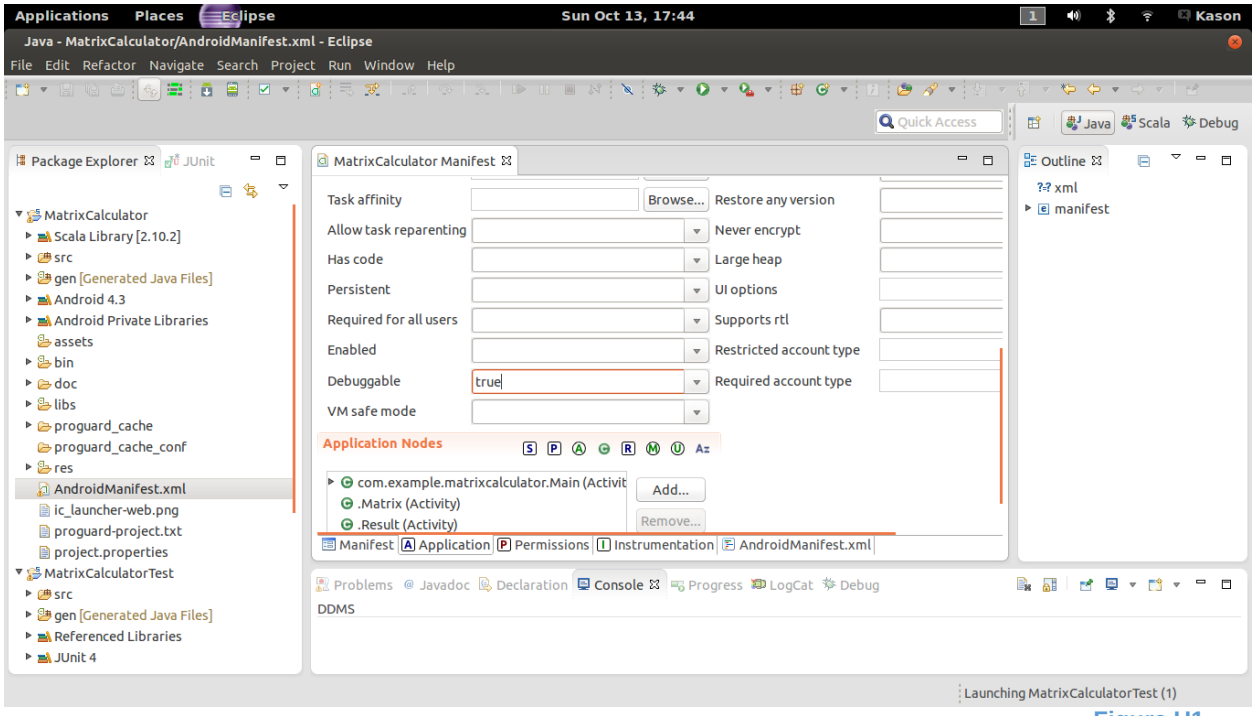


Figure U1

Step 5: Check MatrixCalculatorTest Build Path

- Remove All Android libraries if you see them in MatrixCalculatorTest Build Path.
- As shown in Figure U2 below, MatrixCalculatorTest Build Path should contain only Robotium and JUnit4 libraries.

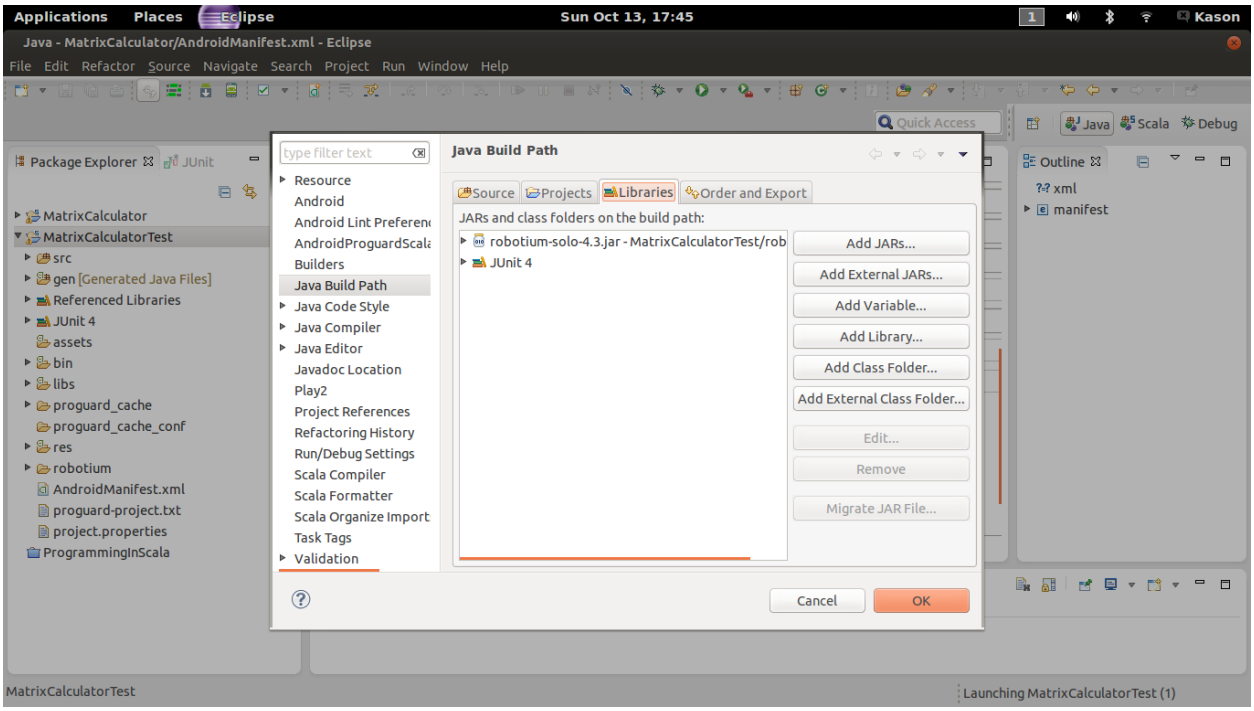


Figure U2

After installing all the prerequisites, Scala, AndroidProguardScala, you can import the source codes folder to Eclipse, compile and execute.

Steps to execute the functional tests:

Step 1: Right click on the `MatrixCalculatorTest` folder in the Package Explorer.
Step 2: Choose Run as `Android JUnit Test`.

Methods implemented with Contracts for Scala

Method 1

```
/**
 * @param x any type value
 * Display the x in String with long length time
 */
def toast(x: Any) = {
  require(x != null) // Precondition
  Toast.makeText(this, x.toString(), Toast.LENGTH_LONG).show()
}
```

The toast method converts the parameter variable `x` of type `Any` into `String` and `toast` (print) with long length time.

- Precondition defines for this method is requiring parameter variable `x` not equals to `null`.
- Expected fault to catch is when the method gets `null` as parameter variable `x`.

Method 2

```
/**
 * @param x the view id value
 * @return the string value of the EditText x
 */
def getTextValue(x: Int): String = {
  require(0 <= x) // Precondition
  findViewById(x).asInstanceOf[EditText].getText().toString()
}
```

The `getTextValue` method get the parameter variable `x` as an `Int`, `findViewByID` as `EditText` and convert the result to `String`.

- Precondition defines for this method is requiring parameter variable `x` is greater or equals to `0`.
- Expected fault to catch when the method get parameter variable `x` that is less than `0`.

Method 3

```
  /**
   * @param x the view id value
   * @return the string value of selected item from the Spinner x
   */
  def getSelectedValue(x: Int): String = {
    require(0 <= x) // Precondition
    String.valueOf(findViewById(x).asInstanceOf[Spinner].getSelectedItem())
  } ensuring (_ != "") // Postcondition
```

The `getSelectedValue` method get the parameter variable `x` as an `Int`, `findViewById` as `Spinner`, gets and returns the `String` value of the selected item.

- Precondition defines for this method is requiring parameter variable `x` is greater or equals to `0`.
- Postcondition defines for this method ensures the result is not equal to **“”** (empty).
- Expected fault to catch when the method get parameter variable `x` that is less than `0`.
- Expected fault to catch when the method return **“”** (empty) result.

Method 4

```
  /**
   * @param x the view id value
   * @param value
   * Set the value to the TextView x
   */
  def setTextValue(x: Int, value: String) = {
    require((0 <= x) && (value != "")) // Precondition
    findViewById(x).asInstanceOf[TextView].setText(value)
  }  ensuring (((!findViewById(x).asInstanceOf[TextView].equals(""))
      || (findViewById(x).asInstanceOf[TextView].equals(value))))
      // Postcondition
```

The `setTextValue` method get the parameter variable `x` as an `Int`, `value` as `String`, `findViewById` using `x` and set the `value` to the `TextView`.

- Precondition defines for this method is requiring parameter variable `x` is greater or equals to `0` and `value` is not equal to **“”** (empty)
- Postcondition defines for this method ensures the result is not equal to **“”** (empty).
- Postcondition defines for this method ensures the result is not equal to the parameter variable `value`.
- Expected fault to catch when the method get parameter variable `x` that is less than `0`.
- Expected fault to catch when the method get parameter variable `value` that is equal to **“”** (empty).
- Expected fault to catch when the textView is equal to **“”** (empty).
- Expected fault to catch when the textView is not equal to parameter variable `value`.

Method 5

```
  /**
   * @param x the view id value
   * Set the view id x to invisible
   */
  def setTextInvisible(x: Int) = {
    require((0 <= x)) // Precondition
    findViewById(x).setVisibility(View.GONE)
  } ensuring (findViewById(x).getVisibility() == View.GONE) // Postcondition
```

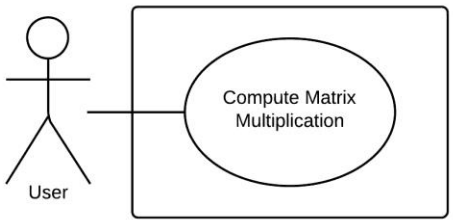
The `setTextInvisible` method get the parameter variable `x` as an `Int`, `findViewById` and `getVisibility` to `GONE`.

- Precondition defines for this method is requiring parameter variable `x` is greater or equals to `0`.
- Postcondition defines the `findViewById(x) getVisibility` is equals to `true`.
- Expected fault to catch when the method get parameter variable `x` that is less than `0`.
- Expected fault to catch when the `findViewById(x) getVisibility` is equals to `View.Gone` is `true`.

Requirement specifications

1. The application allows the user to enter two matrices row size values and two column size values that are to be numeric integer (1, 2, 3, 4, or 5).
2. The application allows the user to enter each matrices row and column size values that are to be one character in length.
3. The column size value of matrix 1 need to be equal to the row size value of matrix 2.
4. The application generates two empty matrices according to the user valid entries of row and column size values.
5. The application allows the user to enter positive or negative integers including numeric characters (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and negative character (-) in the matrices values to the empty matrices values.
6. The application shows the correct result of positive or negative integers after the user entering valid matrices values and application operation.

Use Case Model



Use Cases (UC):

- Compute Matrix Multiplication

Use Case Name: Compute Matrix Multiplication
Summary: The user enters two matrices row and column size values, matrices values and get the result.
Actor: User
Precondition:

- The Matrix Calculator application system is launched.

Description:

1. The user enters two matrices row size values and two column size values.
2. The user selects the Generate button.
3. The system checks the user input sizes values.
4. If the user input sizes values are valid, the system generates and displays the empty matrices.
5. The user enters the matrices values.
6. The user selects the Compute button.
7. The system checks the user input matrices values.
8. If the user input matrices values are valid, the system computes the matrix multiplication.
9. The system displays the result matrix.

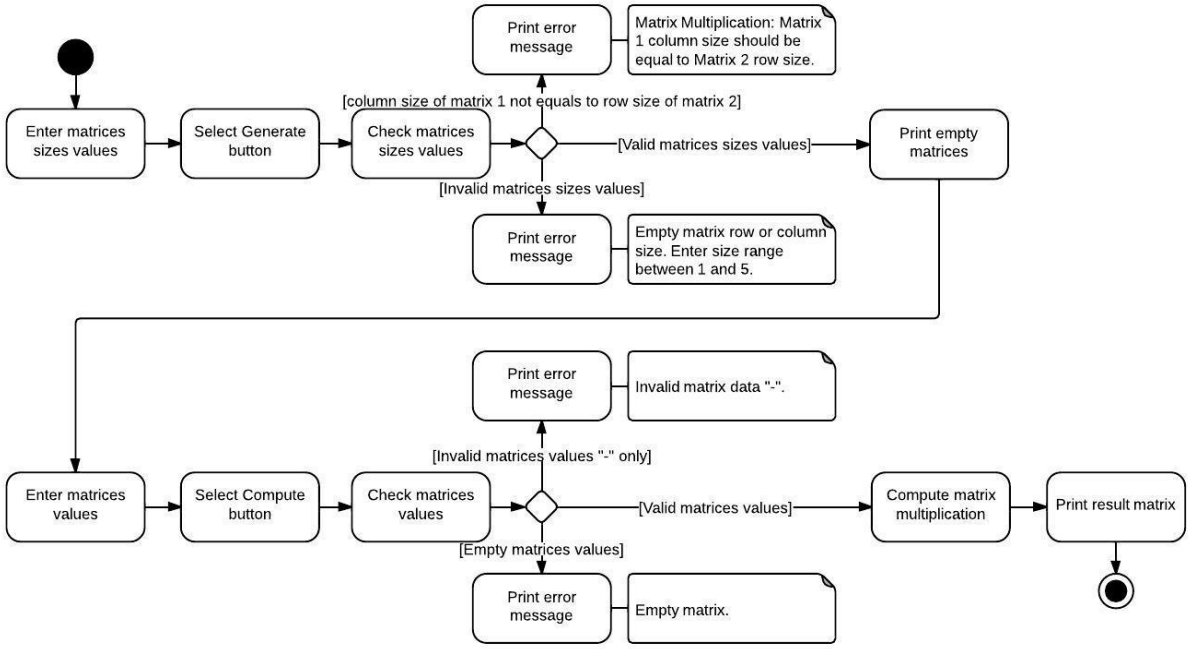
Alternatives:

- If the user select the Generate button after leaving the matrices sizes value(s) empty, the system prints error message “Empty matrix row or column size. Enter size range between 1 and 5.”
- If the user select the Generate button, after inputting size value(s) that is/are not between 1 and 5, the system prints error message “Empty matrix row or column size. Enter size range between 1 and 5.”
- If the user selects the Back button, the application prints the matrices sizes entry page.
- If the user input column size value of matrix 1 is not equal to the row size value of matrix 2, the system prints error message “Matrix Multiplication: Matrix 1 column size should be equal to Matrix 2 row size.”
- If the user select the Compute button after leaving the matrices value(s) empty, the system display error message “Empty matrix.”
- If the user inputs “-” only, the system displays error message "Invalid matrix data "-"."
- If the user input(s) is/are invalid, the system displays error message "Invalid matrix data."
- If the user selects the Back button, the application prints the matrices values entry page.

Postcondition:

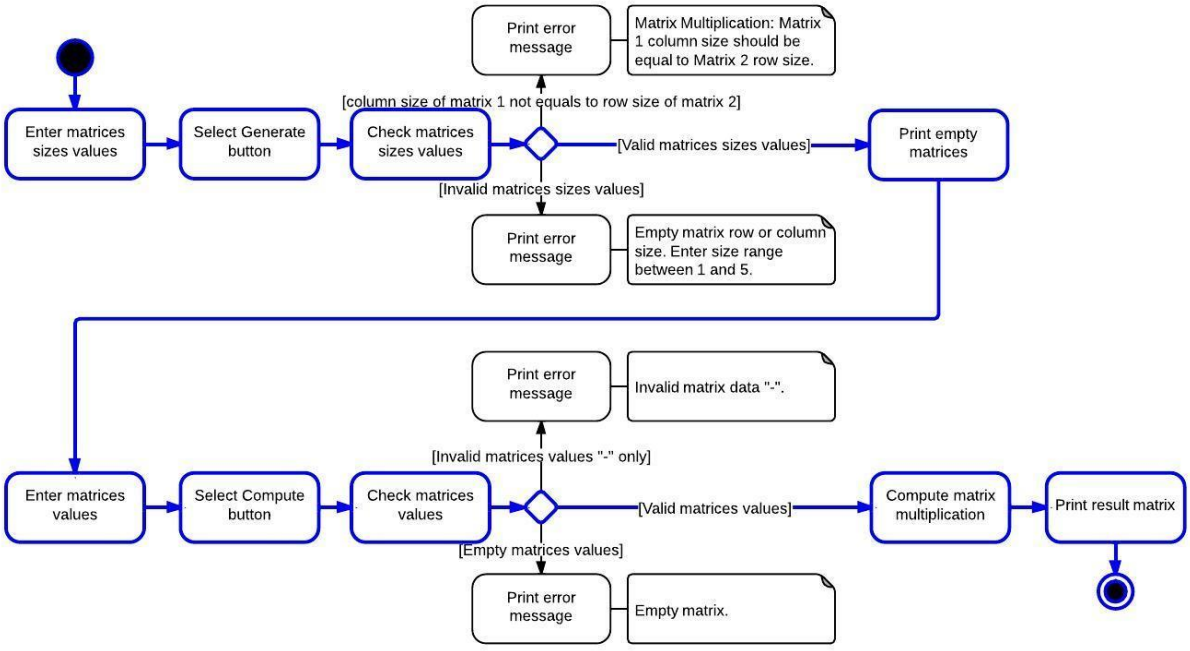
- The result matrix is displayed.

Activity Diagram (AD)

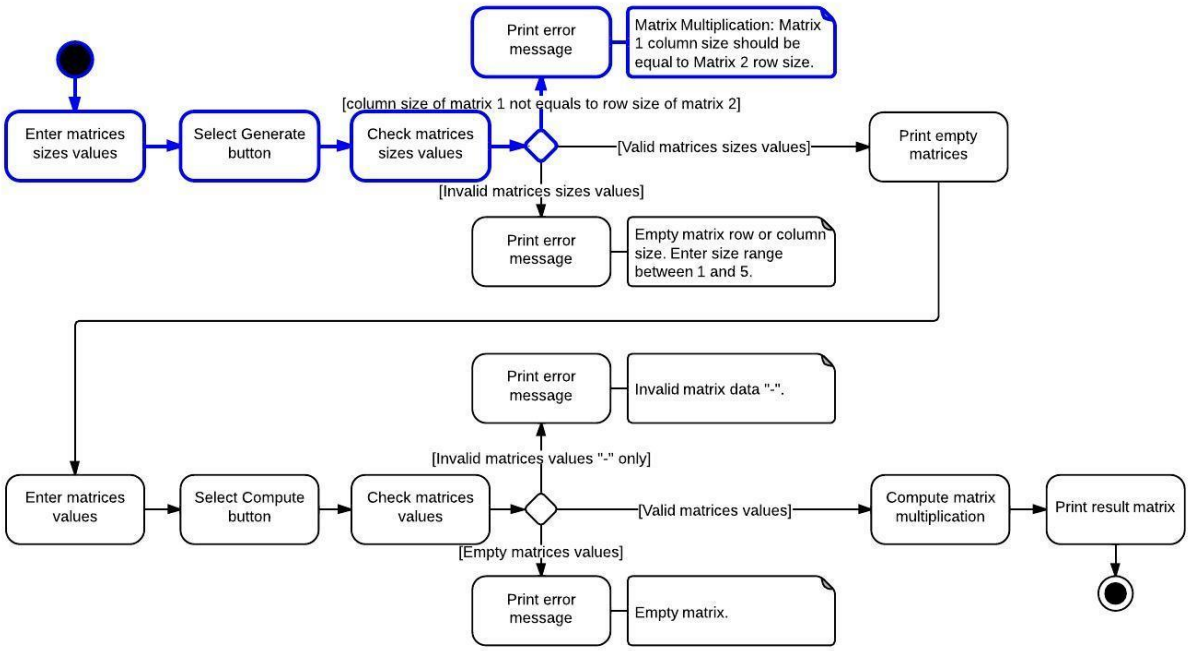


Activity Diagram Test Cases (ADTC)

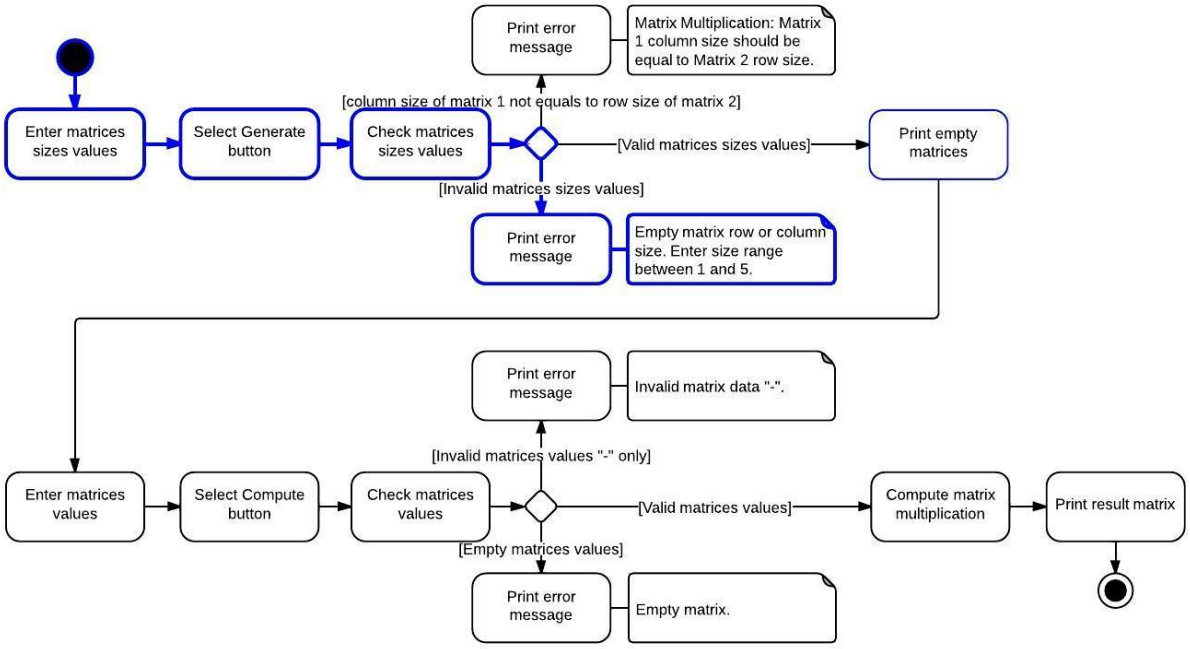
ADTC01



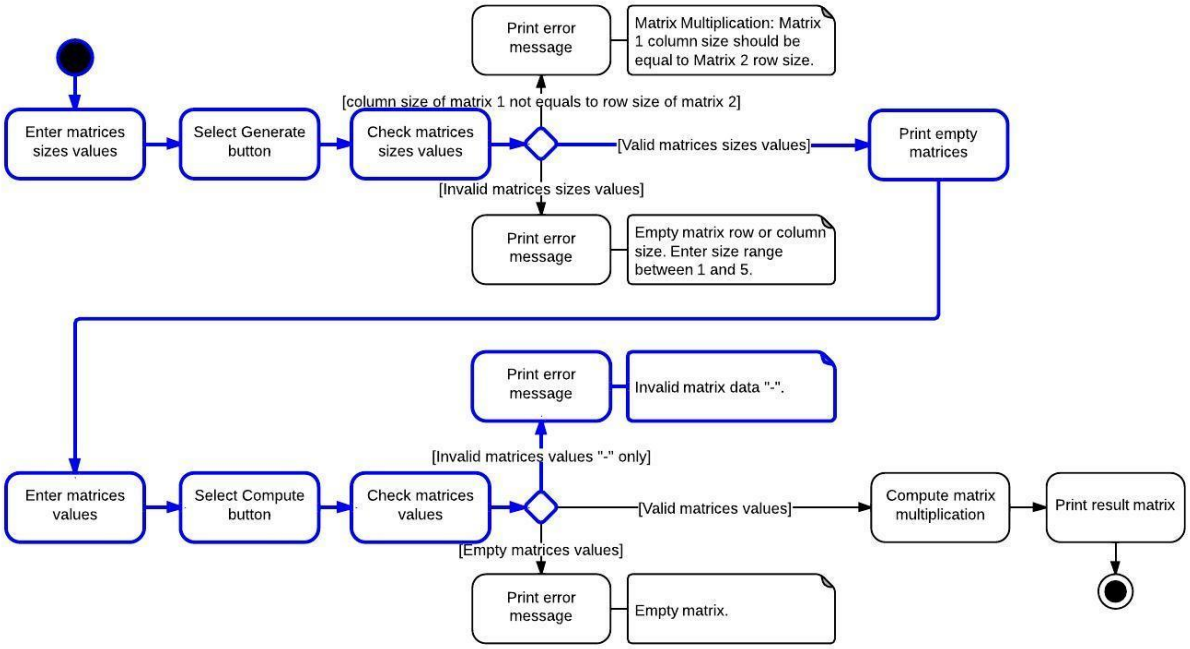
ADTC02



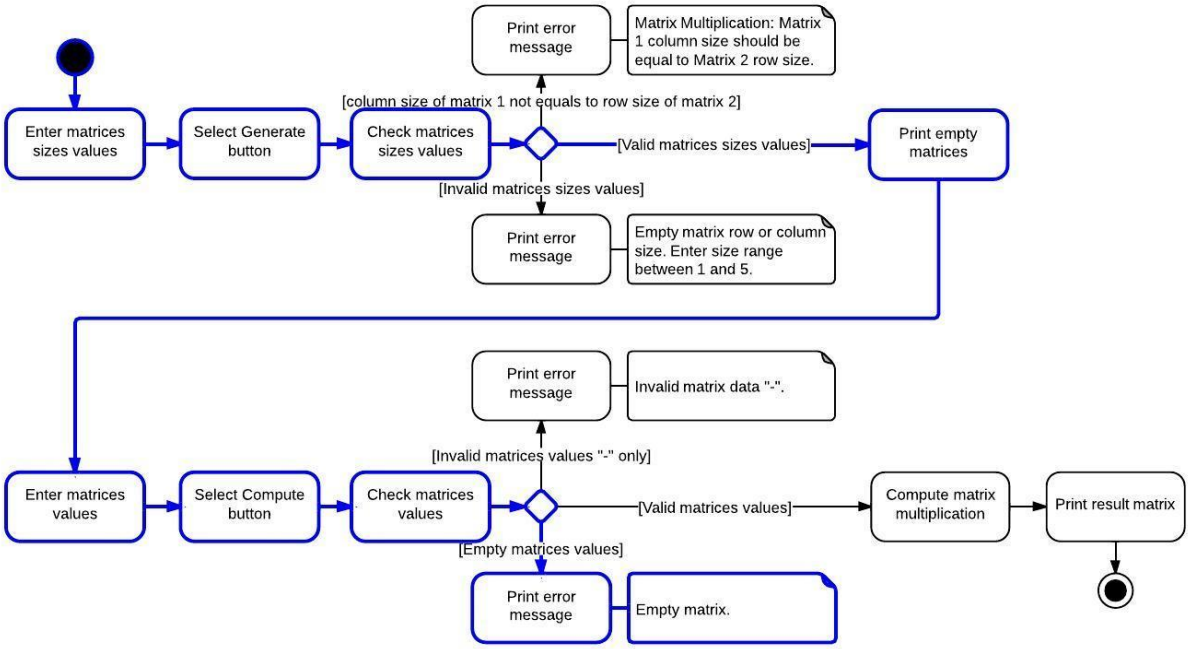
ADTC03



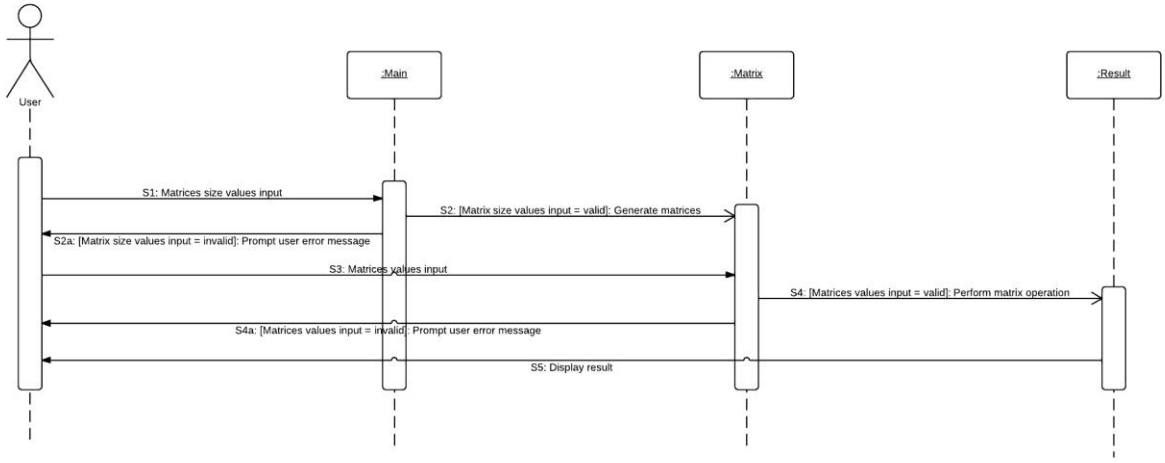
ADTC04



ADTC05

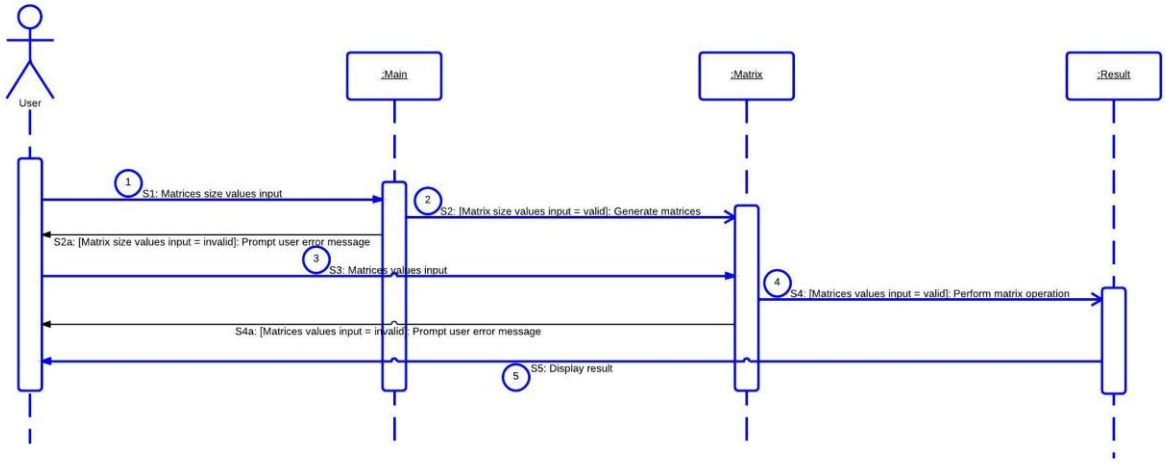


Sequence Diagram (SD)

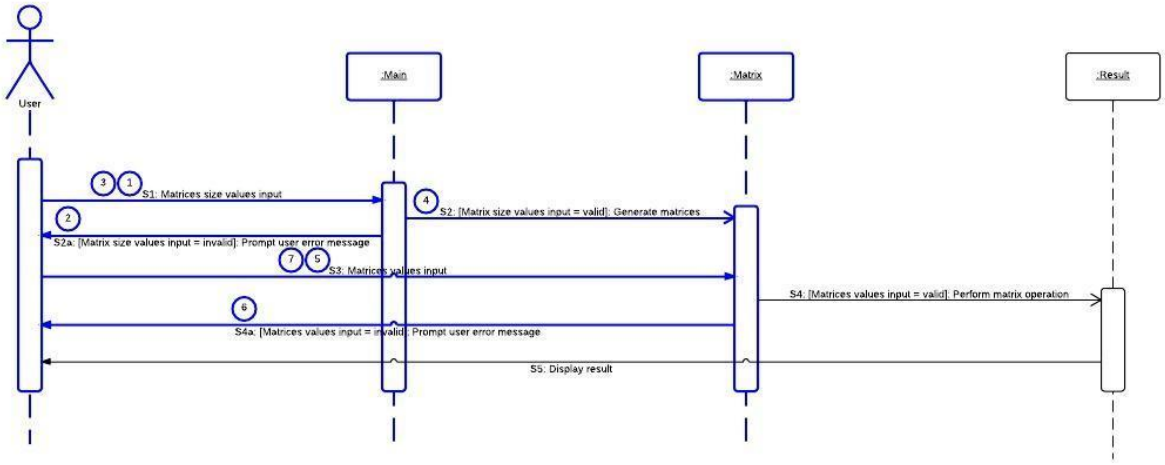


Sequence Diagram Test Cases (SDTC)

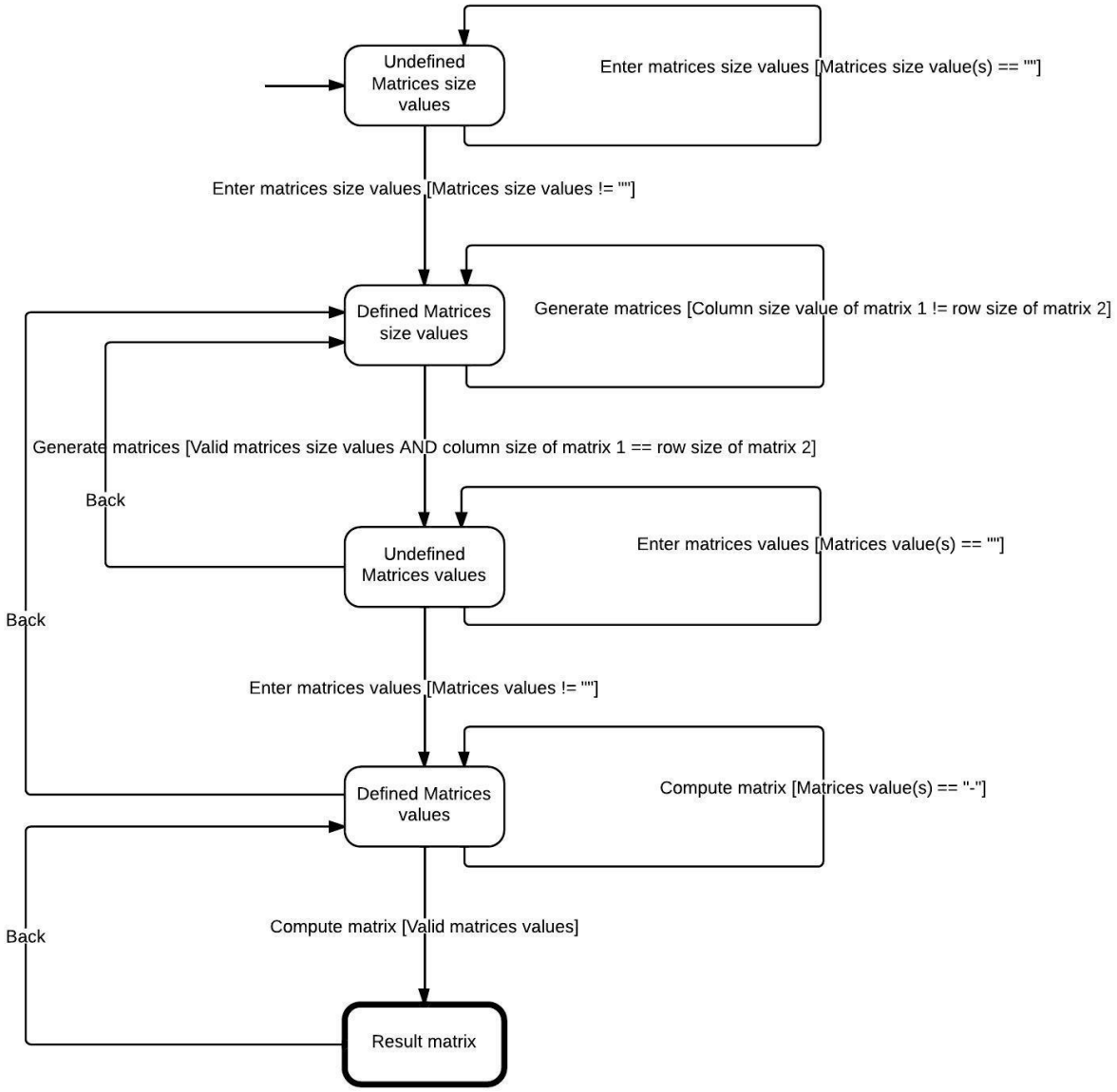
SDTC01



SDTC02

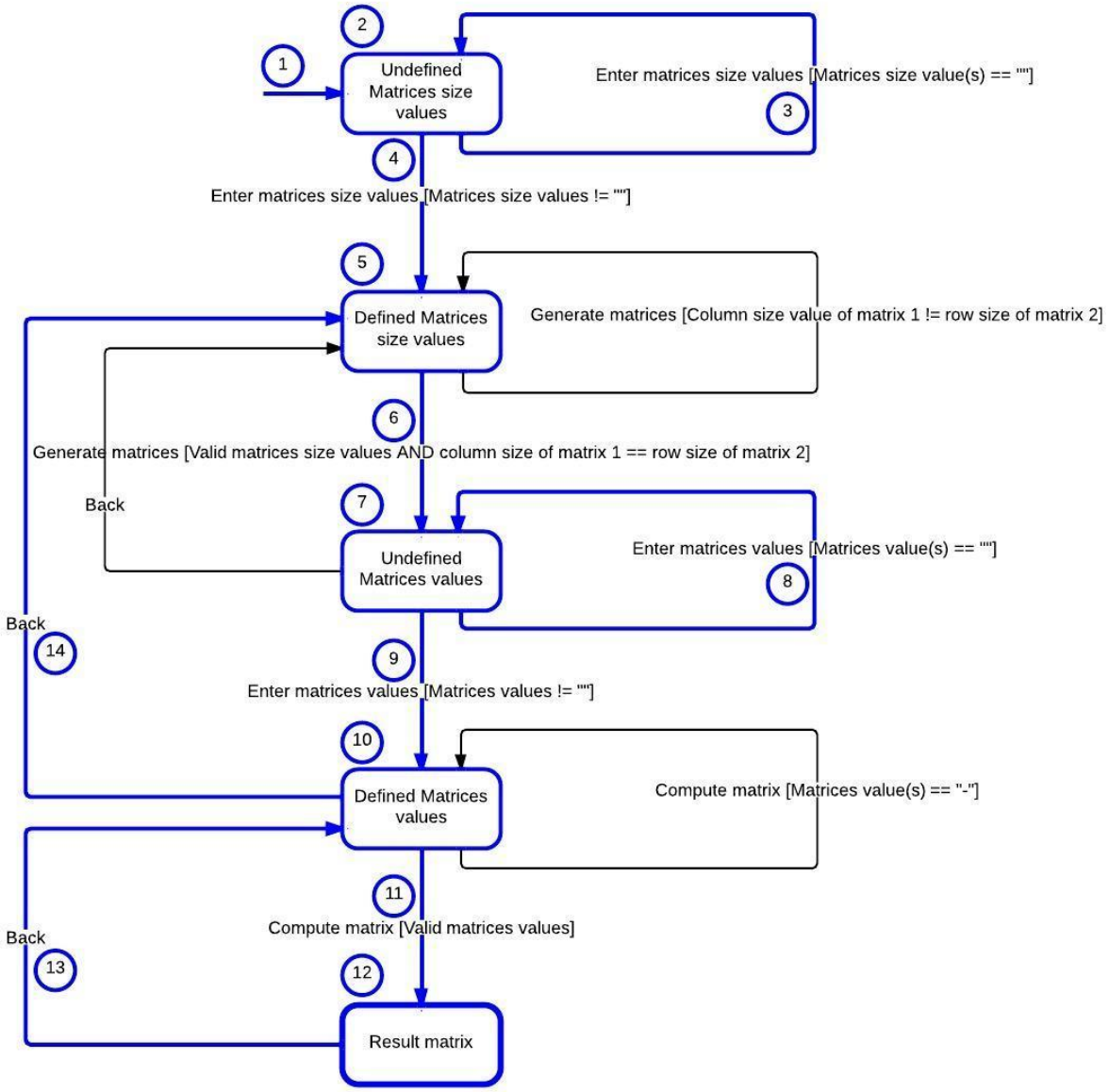

User
:Main
:Matrix
:Result
3
1
S1: Matrices size values input
4
S2: [Matrix size values input = valid]: Generate matrices
2
S2a: [Matrix size values input = invalid]: Prompt user error message
7
5
S3: Matrices values input
S4: [Matrices values input = valid]: Perform matrix operation
6
S4a: [Matrices values input = invalid]: Prompt user error message
S5: Display result

Finite State Machines (FSM)

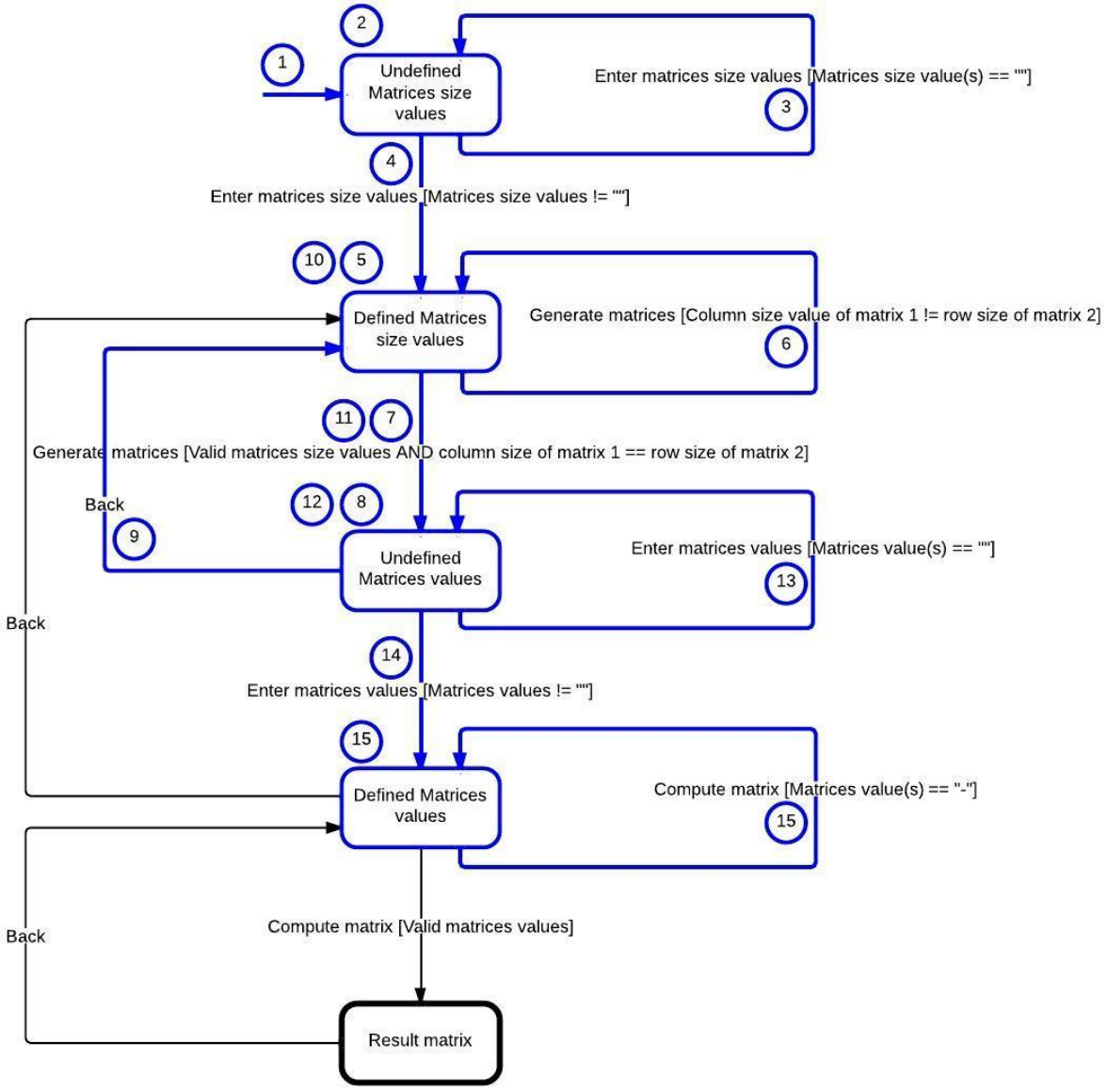


Finite State Machines Test Cases (FSMTC)

FSMTC01



FSMTC02


1
2
Undefined Matrices size values
Enter matrices size values [Matrices size value(s) == ""]
3
4
Enter matrices size values [Matrices size values != ""]
10
5
Defined Matrices size values
Generate matrices [Column size value of matrix 1 != row size of matrix 2]
6
11
7
Generate matrices [Valid matrices size values AND column size of matrix 1 == row size of matrix 2]
Back
12
8
9
Undefined Matrices values
Enter matrices values [Matrices value(s) == ""]
13
Back
14
Enter matrices values [Matrices values != ""]
15
Defined Matrices values
Compute matrix [Matrices value(s) == "-"]
15
Back
Compute matrix [Valid matrices values]
Result matrix

Methods with the all the Test Cases

Method 1
Execution processes are included in `Log1.txt` and `MatrixCalculatorTest 20131128-213751.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Pass
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

Method 2
Execution processes are included in `Log2.txt` and `MatrixCalculatorTest 20131201-135841.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass

Method 2 Cont'd

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Pass
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

Method 3

Execution processes are included in `Log3.txt` and `MatrixCalculatorTest 20131201-143824.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass

Method 3 Cont'd

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Fail
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

Method 4

Execution processes are included in `Log4.txt` and `MatrixCalculatorTest 20131201-144605.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Fail

Method 4 Cont'd

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

Method 5

Execution processes are included in `Log5.txt` and `MatrixCalculatorTest 20131201-150728.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Pass
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

Method 1-5 Combined
Execution processes are included in `Log1_5.txt` and `MatrixCalculatorTest 20131201-151419.xml`

Test Case #	Test Input	Expected Output	Comments	Pass/Fail
ADTC01	1, 1, 1, 1 1, 2	2		Pass
ADTC02	1, 1, 2, 1	Error message		Pass
ADTC03	1, 1, 1, 0	Error message		Pass
ADTC04	1, 1, 1, 1 1, -	Error message		Pass
ADTC05	1, 1, 1, 1 1,	Error message		Pass
FSMTC01	1, 1, 1, 1 3, 4	12		Pass
FSMTC02	1, 1, 2, 1 1, 1, 1, 1 3, -	Error message		Fail
SDTC01	1, 1, 1, 1 3, 4	12		Pass
SDTC02	1, 2, 1, 1 1, 1, 1, 1 3, -	Error message		Pass

References

- Scala - Design by Contract http://blog.m1key.me/2010/02/programming-scala-design-by-contract.html
- Contracts for Scala: http://link.springer.com/chapter/10.1007%2F978-3-642-16612-9_5#page-2