UNIVERSITÉ Concordia UNIVERSITY

SOEN 6011
SOFTWARE ENGINEERING PROCESSES

DELIVERABLE 3

ETERNITY : FUNCTIONS

SUBMITTED BY:
Prabhpreet Singh

DVCS Url :
`https://github.com/prabhpreet429/SOEN6011`

August 3, 2019

Contents

List of Figures

List of Tables

1 Problem 5

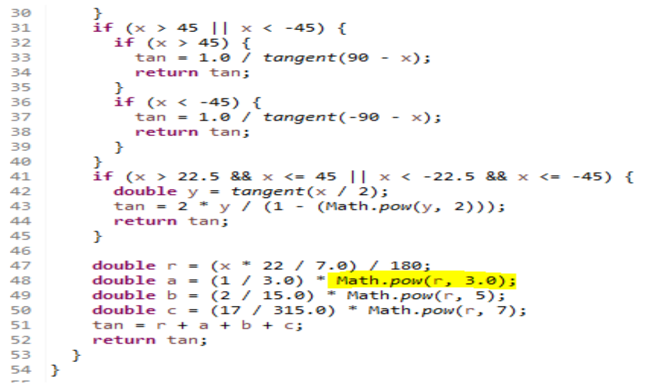
1.1 Source Code Review

This is the source code review of F2: $tan(x)$ function.

1.1.1 Constraints

The constraints are checked for the code manually.

- **Build from scratch:** The code violates this constraint as the program has used many in-built functions such as Math.pow(), String length function etc.



```
    }
    if (x > 45 || x < -45) {
      if (x > 45) {
        tan = 1.0 / tangent(90 - x);
        return tan;
      }
      if (x < -45) {
        tan = 1.0 / tangent(-90 - x);
        return tan;
      }
    }
    if (x > 22.5 && x <= 45 || x < -22.5 && x <= -45) {
      double y = tangent(x / 2);
      tan = 2 * y / (1 - (Math.pow(y, 2)));
      return tan;
    }

    double r = (x * 22 / 7.0) / 180;
    double a = (1 / 3.0) * Math.pow(r, 3.0);
    double b = (2 / 15.0) * Math.pow(r, 5);
    double c = (17 / 315.0) * Math.pow(r, 7);
    tan = r + a + b + c;
    return tan;
  }
}
```

Figure 1: Using in-built library functions

- **Identical programming style across the team:** The code follows this constraint as it follows the common programming style by google.

- **The program should compile:** The code compiles and gives result.

1.1.2 Functionality

The functionality of the code is checked manually. The results are checked against the actual value.

- **Implement functionality:** The code provides the solution for $tan(x)$ and it gives the correct output. Hence it implements the required functionality.

- The result of the implementation is correct at the middle of the interval of $-\pi/2$ to $\pi/2$. As the input value goes close to $\pi/2$ or close to $-\pi/2$, the deviation in the result is seen. For example, the value of $tan(89.9) = 572.9572$, but the result shows 572.73 which has absolute error of 0.22.

- The result of implementation gives the value round off to two decimal places.

1.1.3 Graphical User Interface

The Graphical User Interface is checked manually to see its usability for the user.

- **Working interface:** The buttons and fields used in the interface are working.
- The GUI doesn't specify the units in which the user has to enter the value. It doesn't specify the units of the output result.

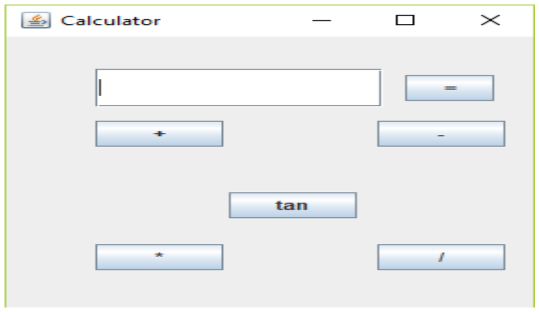


Figure 2: No marking of Units

1.1.4 General

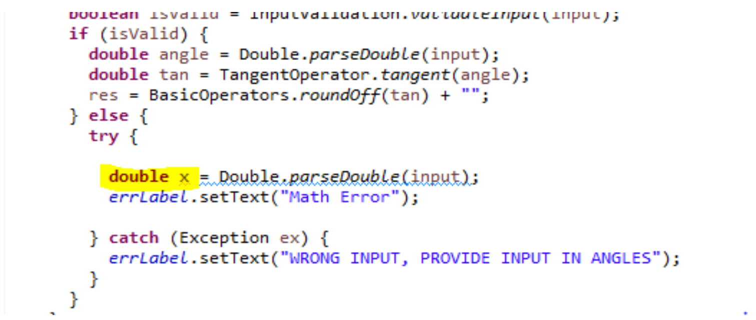
Manual Code Checking

- **Check syntax errors:** There are no syntax errors or grammatical errors found in the code.
- **Use of Meaningful Variable, Method and Names:** Meaningful names are not used in the code. Short variable names such as ”x” and ”p” are used which is not good for code.
- **Naming Convention:** The names used for classes, methods, variables are according to the google programming style.
- **Code Formatting:** The code formatting is applied in the code. The braces, indentation, whitespaces are properly coded.
- **Proper documentation:** Proper documentation of each method is done but not for class.
- **Comments are Required:** Comments are not seen in the code for any field or variable.

Automatic Code Checking
Tool Used: PMD

- **Don't Repeat Code, Use Functions instead:** The code has used functions and so there is no repetition of code found.
- **Use Exceptions rather than Return codes:** The Exceptions are used well to give error messages in the code.

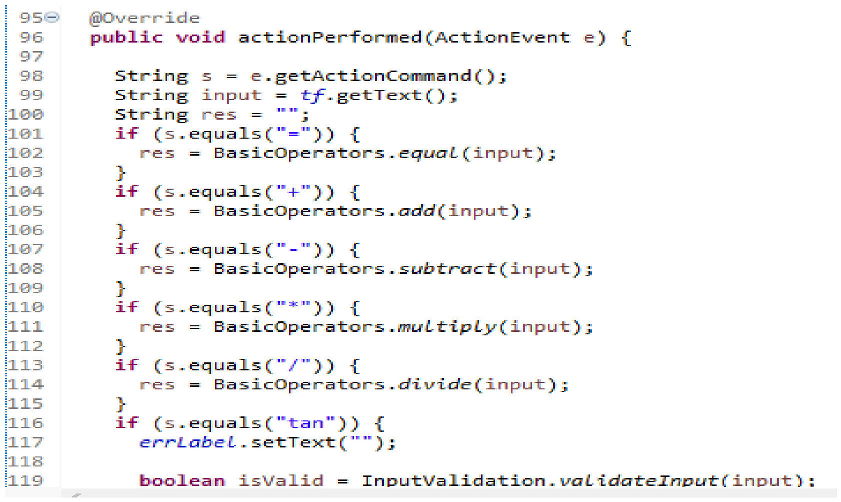
- **Remove needless statements such as for testing purposes:** The useless assignment of $double\ tan = 0.0$ in class TangentOperator.java is not required. Rest no other useless statements are found in the code.
- **Don't return Null:** No null is returned in the code.
- **Don't write secured information directly:** No secured information such as passwords are written directly in the code.
- **Remove Useless parenthesis:** A statement in class TangentOperator.java is $x = x-(((int)x/180))*180$;. This statement has useless parenthesis which can be removed without affecting the calculation.
- **Don't create unused variables:** Unused variables are found in the code. The variable x in the image below is unused.



```
boolean isValid = InputValidation.validateInput(input);
if (isValid) {
  double angle = Double.parseDouble(input);
  double tan = TangentOperator.tangent(angle);
  res = BasicOperators.roundOff(tan) + "";
} else {
  try {

    double x = Double.parseDouble(input);
    errLabel.setText("Math Error");

  } catch (Exception ex) {
    errLabel.setText("WRONG INPUT, PROVIDE INPUT IN ANGLES");
  }
}
```

Figure 3: Unused Variable

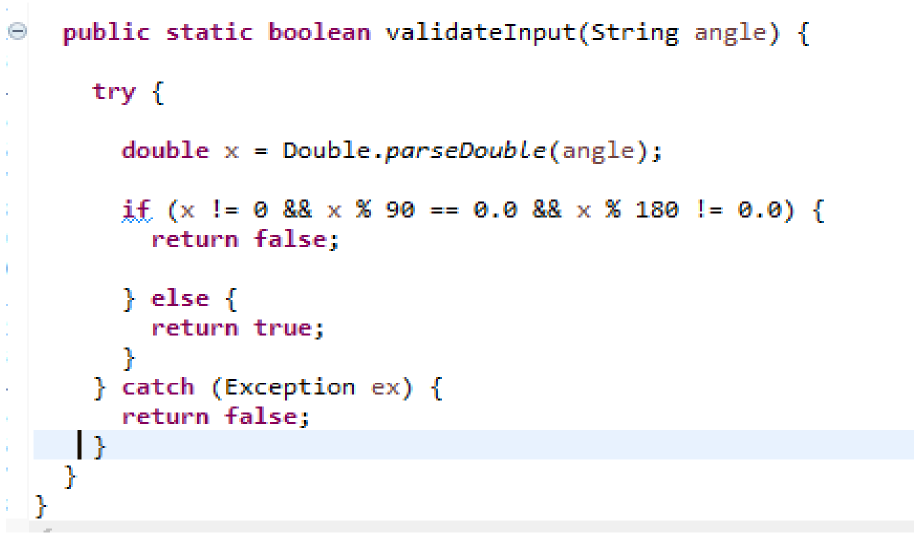
- **Data Flow Analysis:** The data flow anomaly DD occurs here because values are assigned to variables more than one time. The later assignments will always overwrite the previous values. For example in the diagram below, variable res is assigned values continuously.



```
@Override
public void actionPerformed(ActionEvent e) {

  String s = e.getActionCommand();
  String input = tf.getText();
  String res = "";
  if (s.equals("=")) {
    res = BasicOperators.equal(input);
  }
  if (s.equals("+")) {
    res = BasicOperators.add(input);
  }
  if (s.equals("-")) {
    res = BasicOperators.subtract(input);
  }
  if (s.equals("*")) {
    res = BasicOperators.multiply(input);
  }
  if (s.equals("/")) {
    res = BasicOperators.divide(input);
  }
  if (s.equals("tan")) {
    errLabel.setText("");

    boolean isValid = InputValidation.validateInput(input);
```

Figure 4: Data Flow Anomaly

- **Only One Return:** The method should give only one return statement. For example in the figure below there are multiple return statements.

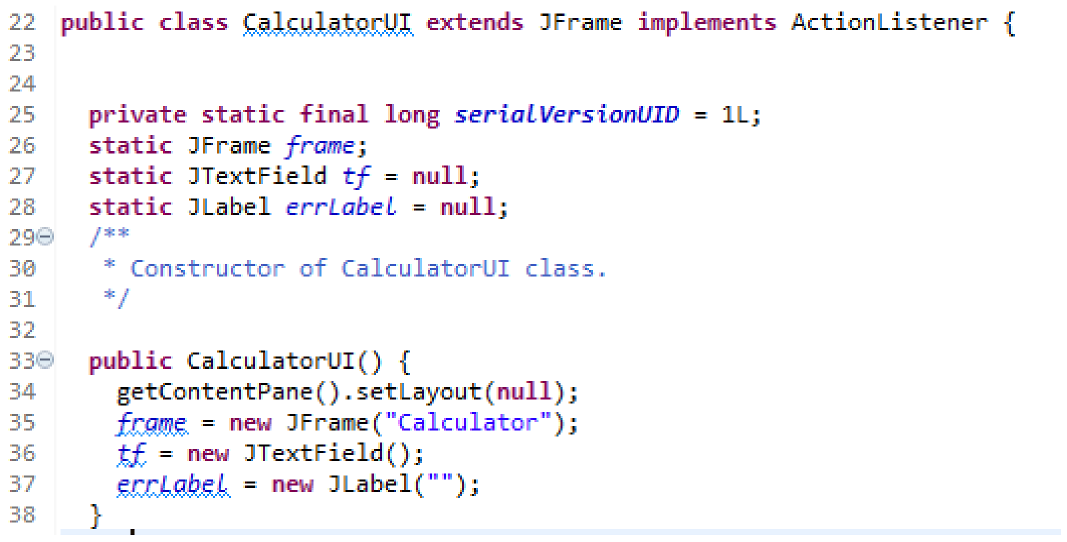


```java
public static boolean validateInput(String angle) {

  try {

    double x = Double.parseDouble(angle);

    if (x != 0 && x % 90 == 0.0 && x % 180 != 0.0) {
      return false;

    } else {
      return true;
    }
  } catch (Exception ex) {
    return false;
  }
 }
}
```

Figure 5: Only One Return

- **Avoid Catching General Exceptions:** In the above figure, it is also seen that exceptions are caught by general Exception class. Specific Exceptions must be caught which are specific to the errors such as NullPointerException, NumberFormatException, RunTime Exception.

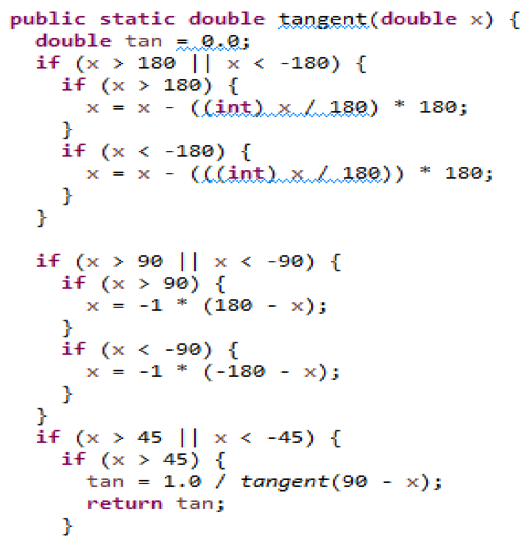
- **Assigning a value to a static field in a constructor:** The static field should not be updated in constructor. It is being updated in the code as shown below.



```java
public class CalculatorUI extends JFrame implements ActionListener {


  private static final long serialVersionUID = 1L;
  static JFrame frame;
  static JTextField tf = null;
  static JLabel errLabel = null;
  /**
   * Constructor of CalculatorUI class.
   */

  public CalculatorUI() {
    getContentPane().setLayout(null);
    frame = new JFrame("Calculator");
    tf = new JTextField();
    errLabel = new JLabel("");
  }
```

Figure 6: Static fields should not be updated in constructor

- **Avoid Reassigning of Parameters:** If the parameters in a method are reassigned values, then the original value is lost. So the parameters should not be assigned any values. For example, in the code given below, the value of parameter x is reassigned. Other way is to use another variable and use the value of x.



```java
public static double tangent(double x) {
  double tan = 0.0;
  if (x > 180 || x < -180) {
    if (x > 180) {
      x = x - ((int) x / 180) * 180;
    }
    if (x < -180) {
      x = x - (((int) x / 180)) * 180;
    }
  }

  if (x > 90 || x < -90) {
    if (x > 90) {
      x = -1 * (180 - x);
    }
    if (x < -90) {
      x = -1 * (-180 - x);
    }
  }
  if (x > 45 || x < -45) {
    if (x > 45) {
      tan = 1.0 / tangent(90 - x);
      return tan;
    }
```

Figure 7: Avoid Reassigning Parameters

1.1.5 Approach for Automatic Code Review

A Code Review Tool PMD is used to check the code. PMD is a tool for source code analysis. It is used to find bugs in the code such as unused code, unreachable code, unnecessary statements in code and so on. It is installed in eclipse as plug-in. It is run by Right clicking on project $- >$ PMD $- >$ Check Code. It generates a report for code violations.

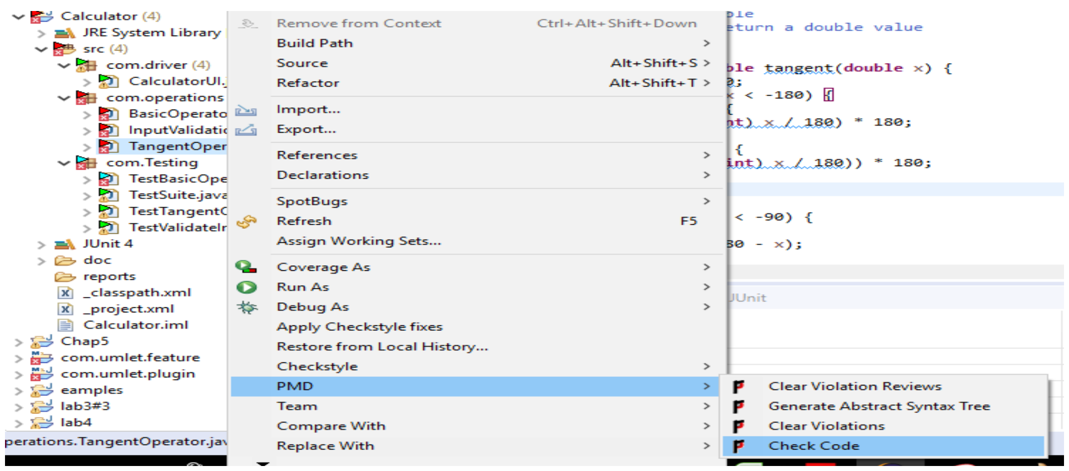
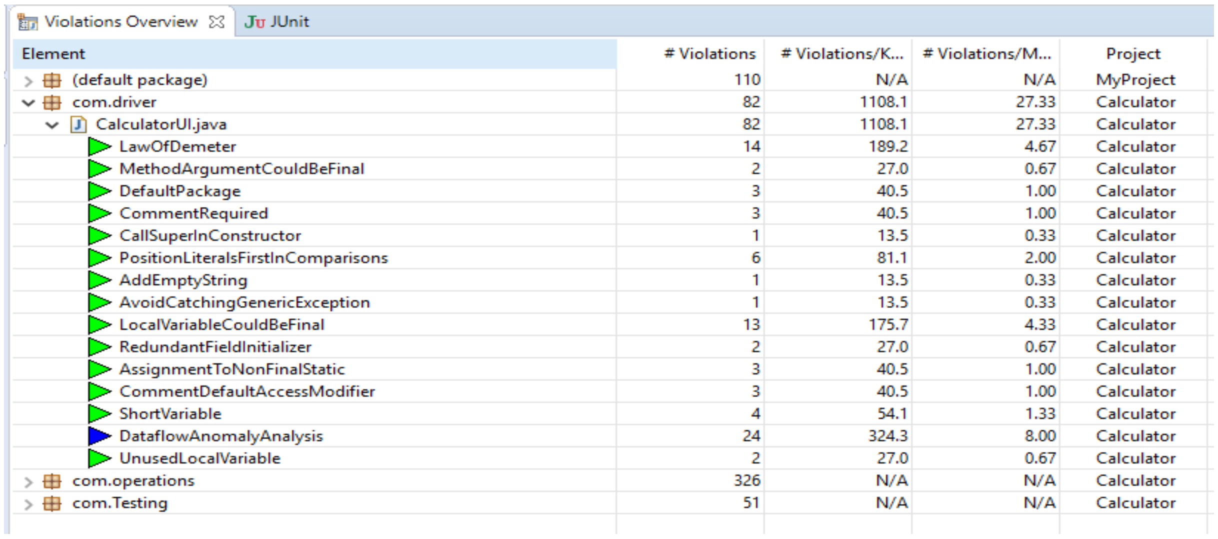


Figure 8: Using PMD

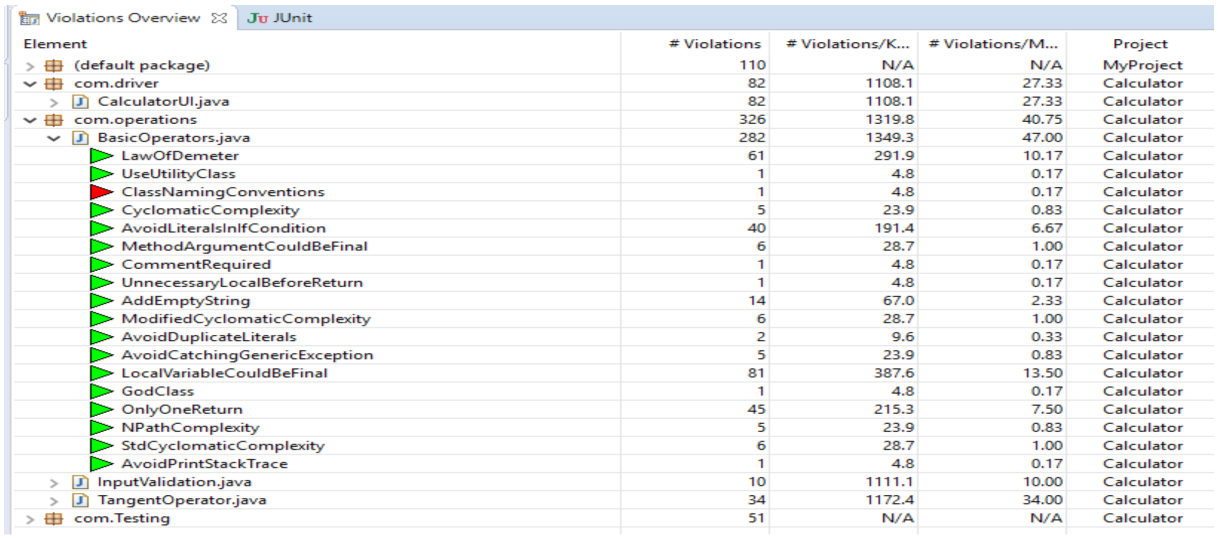
Violation Report of CalculatorUI.java



Element	# Violations	# Violations/K...	# Violations/M...	Project
(default package)	110	N/A	N/A	MyProject
com.driver	82	1108.1	27.33	Calculator
CalculatorUI.java	82	1108.1	27.33	Calculator
LawOfDemeter	14	189.2	4.67	Calculator
MethodArgumentCouldBeFinal	2	27.0	0.67	Calculator
DefaultPackage	3	40.5	1.00	Calculator
CommentRequired	3	40.5	1.00	Calculator
CallSuperInConstructor	1	13.5	0.33	Calculator
PositionLiteralsFirstInComparisons	6	81.1	2.00	Calculator
AddEmptyString	1	13.5	0.33	Calculator
AvoidCatchingGenericException	1	13.5	0.33	Calculator
LocalVariableCouldBeFinal	13	175.7	4.33	Calculator
RedundantFieldInitializer	2	27.0	0.67	Calculator
AssignmentToNonFinalStatic	3	40.5	1.00	Calculator
CommentDefaultAccessModifier	3	40.5	1.00	Calculator
ShortVariable	4	54.1	1.33	Calculator
DataflowAnomalyAnalysis	24	324.3	8.00	Calculator
UnusedLocalVariable	2	27.0	0.67	Calculator
com.operations	326	N/A	N/A	Calculator
com.Testing	51	N/A	N/A	Calculator

Figure 9: Violation Report of CalculatorUI.java

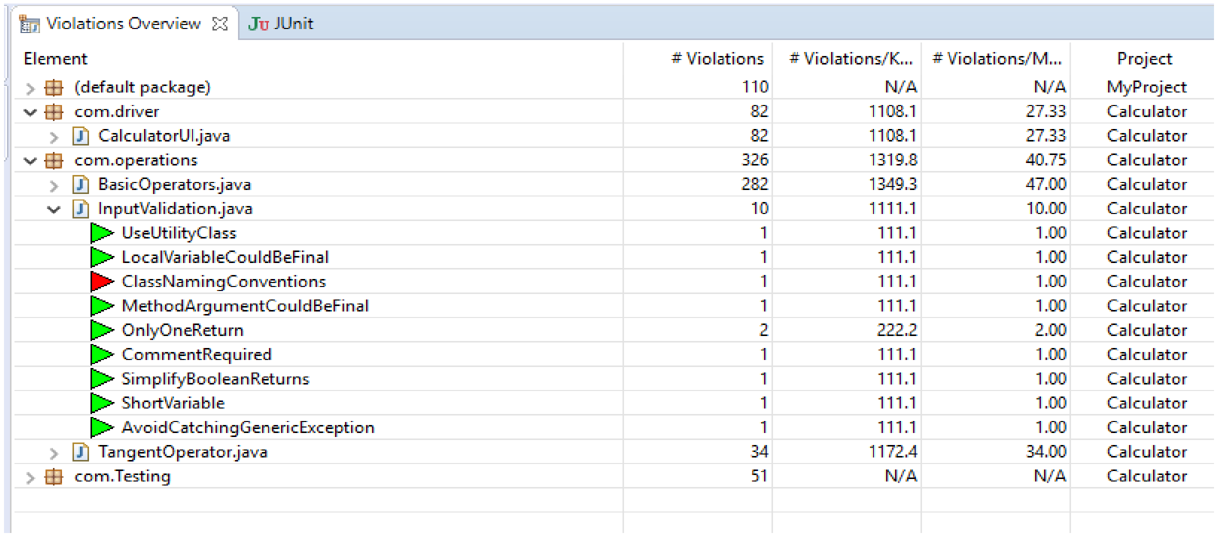
Violation Report of BasicOperations.java



Element	# Violations	# Violations/K...	# Violations/M...	Project
(default package)	110	N/A	N/A	MyProject
com.driver	82	1108.1	27.33	Calculator
CalculatorUI.java	82	1108.1	27.33	Calculator
com.operations	326	1319.8	40.75	Calculator
BasicOperators.java	282	1349.3	47.00	Calculator
LawOfDemeter	61	291.9	10.17	Calculator
UseUtilityClass	1	4.8	0.17	Calculator
ClassNamingConventions	1	4.8	0.17	Calculator
CyclomaticComplexity	5	23.9	0.83	Calculator
AvoidLiteralsInIfCondition	40	191.4	6.67	Calculator
MethodArgumentCouldBeFinal	6	28.7	1.00	Calculator
CommentRequired	1	4.8	0.17	Calculator
UnnecessaryLocalBeforeReturn	1	4.8	0.17	Calculator
AddEmptyString	14	67.0	2.33	Calculator
ModifiedCyclomaticComplexity	6	28.7	1.00	Calculator
AvoidDuplicateLiterals	2	9.6	0.33	Calculator
AvoidCatchingGenericException	5	23.9	0.83	Calculator
LocalVariableCouldBeFinal	81	387.6	13.50	Calculator
GodClass	1	4.8	0.17	Calculator
OnlyOneReturn	45	215.3	7.50	Calculator
NPathComplexity	5	23.9	0.83	Calculator
StdCyclomaticComplexity	6	28.7	1.00	Calculator
AvoidPrintStackTrace	1	4.8	0.17	Calculator
InputValidation.java	10	1111.1	10.00	Calculator
TangentOperator.java	34	1172.4	34.00	Calculator
com.Testing	51	N/A	N/A	Calculator

Figure 10: Violation Report of BasicOperations.java

Violation Report of InputValidation.java

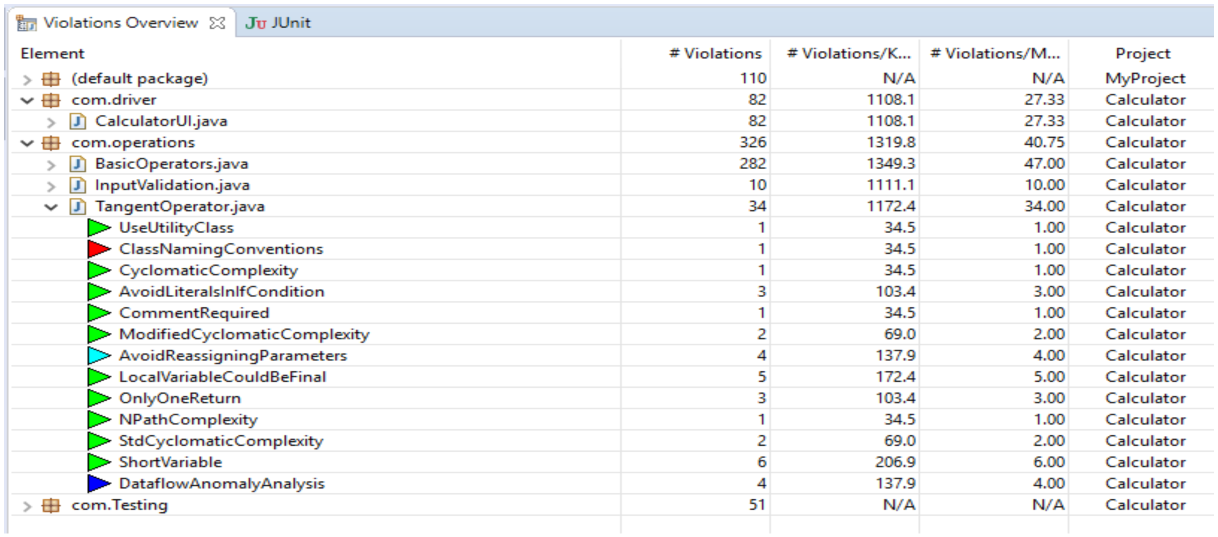


Violations Overview | JUnit

Element	# Violations	# Violations/K...	# Violations/M...	Project
(default package)	110	N/A	N/A	MyProject
com.driver	82	1108.1	27.33	Calculator
CalculatorUI.java	82	1108.1	27.33	Calculator
com.operations	326	1319.8	40.75	Calculator
BasicOperators.java	282	1349.3	47.00	Calculator
InputValidation.java	10	1111.1	10.00	Calculator
UseUtilityClass	1	111.1	1.00	Calculator
LocalVariableCouldBeFinal	1	111.1	1.00	Calculator
ClassNamingConventions	1	111.1	1.00	Calculator
MethodArgumentCouldBeFinal	1	111.1	1.00	Calculator
OnlyOneReturn	2	222.2	2.00	Calculator
CommentRequired	1	111.1	1.00	Calculator
SimplifyBooleanReturns	1	111.1	1.00	Calculator
ShortVariable	1	111.1	1.00	Calculator
AvoidCatchingGenericException	1	111.1	1.00	Calculator
TangentOperator.java	34	1172.4	34.00	Calculator
com.Testing	51	N/A	N/A	Calculator

Figure 11: Violation Report of InputValidation.java

Violation Report of TangentOperator.java



Violations Overview | JUnit

Element	# Violations	# Violations/K...	# Violations/M...	Project
(default package)	110	N/A	N/A	MyProject
com.driver	82	1108.1	27.33	Calculator
CalculatorUI.java	82	1108.1	27.33	Calculator
com.operations	326	1319.8	40.75	Calculator
BasicOperators.java	282	1349.3	47.00	Calculator
InputValidation.java	10	1111.1	10.00	Calculator
TangentOperator.java	34	1172.4	34.00	Calculator
UseUtilityClass	1	34.5	1.00	Calculator
ClassNamingConventions	1	34.5	1.00	Calculator
CyclomaticComplexity	1	34.5	1.00	Calculator
AvoidLiteralsInIfCondition	3	103.4	3.00	Calculator
CommentRequired	1	34.5	1.00	Calculator
ModifiedCyclomaticComplexity	2	69.0	2.00	Calculator
AvoidReassigningParameters	4	137.9	4.00	Calculator
LocalVariableCouldBeFinal	5	172.4	5.00	Calculator
OnlyOneReturn	3	103.4	3.00	Calculator
NPathComplexity	1	34.5	1.00	Calculator
StdCyclomaticComplexity	2	69.0	2.00	Calculator
ShortVariable	6	206.9	6.00	Calculator
DataflowAnomalyAnalysis	4	137.9	4.00	Calculator
com.Testing	51	N/A	N/A	Calculator

Figure 12: Violation Report of TangentOperator.java

2 Problem 7

2.1 Implementing Test Cases

Testing the function F3: $sinh(x)$. The sinh(x) is the hyperbolic sin function. This function is defined as $e^x - e^{-x/2}$.
TestValidator.java: This class is used to test if the input value entered by the user is valid for implementaton or not. For validation, certain checks are made.

Test Cases	**Review**
Test Case 1: Not Null	The string ”123” is checked if it is null or not. The string implemented is not null which is true. It passes the test. There is no test where empty string is passed to check if it is null.
Test Case 2: Not Alpha	The string ”123” is tested to see if it contains alphabets or not. The string implemented doesn't contain alphabets which is true. It passes the test. The string ”12a” is tested. It gives false value because it contains alphabets. The test is passed.
Test Case 3: No Space	The string ”123” is tested to see if it contains space or not. The string implemented doesn't contain space which is true. It passes the test. The string ”12 3” is tested. It gives false value because it contains space. The test is passed.
Test Case 4: No Special Characters	The string ”123” is tested to see if it contains special characters or not. The string implemented doesn't contain space which is true. It passes the test. The string ”1#523” is tested. It gives false value because it contains special characters. The test is passed.
Test Case 5: Testing Validation Function	The string ”123” is tested to see if it does not contain any space, alphabets, special characters or is not null. The string implemented passes the test. The string ”123a” is tested. It gives false value because it contains alphabets. The test is passed.

Table 1: TestValidator Test Cases

TestMyFuncs.java: This class is used to test the supporting functions that are to be used in the implementation of main function.

Test Cases	**Review**
Test Case 1: String Length Function	The function is used to return the length of a string. If the input is ”sf”, output comes out to be 2. The test is passed.
Test Case 2: Exponential Function	The exponential(int n, double x) function is used to find the exponential value at nth term of value x. The value tested is $n = 20$ and $x = 5$ which gives a value 148.413 which is correct. It passes the test.

Table 2: TestMyFuncs Test Cases

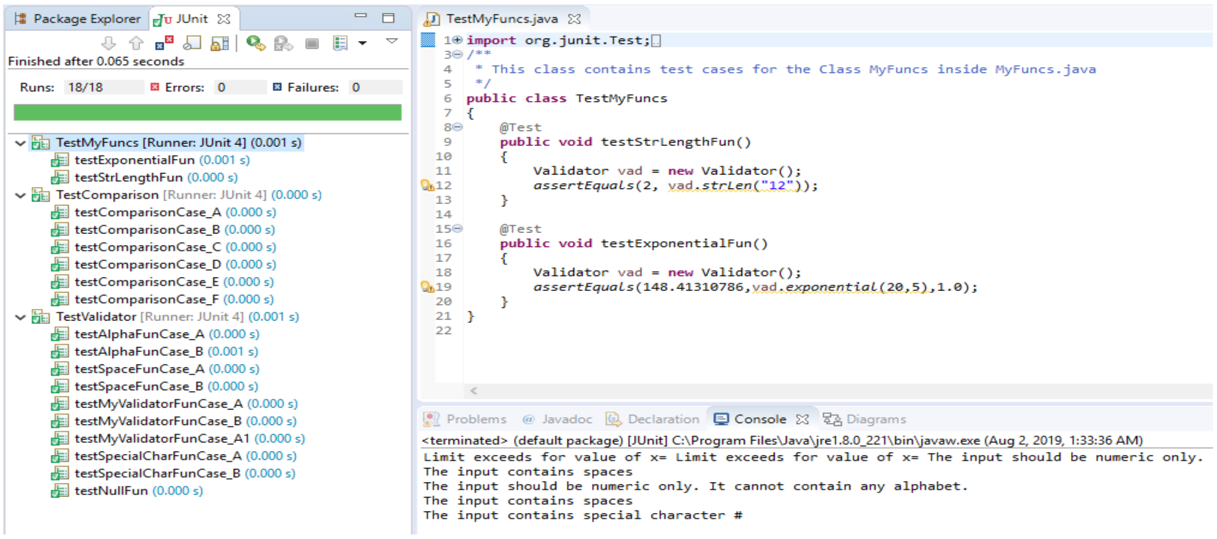
TestComparison.java: This class tests the main implementation of the function $sinh(x)$.

Test Cases	**Review**
Test Case 1: $x = 123$	The test is run on double value 123 to calculate its sinh. The value is passed to comparison method and returns 1.3097586593E53 which is the correct value of $sinh(123)$. Hence the test passes.
Test Case 2: $x = 0$	The test is run on value 0 to calculate its $sinh$. The value is passed to comparison method and returns 0 which is the correct value of $sinh(0)$. Hence the test passes.
Test Case 3: $x = 1000$	The test is run on double value 1000 to calculate its $sinh$. The value is passed to comparison method and returns 1000 along with error message ”Limit exceeds for value of x=”. The test should not return 1000 because 1000 is not the correct answer for $sinh(1000)$.
Test Case 4: $x = -1000$	The test is run on double value -1000 to calculate its $sinh$. The value is passed to comparison method and returns -1000 along with error message ”Limit exceeds for value of x=”. The test should not return -1000 because -1000 is not the correct answer for $sinh(-1000)$.
Test Case 5: $x = -123$	The test is run on double value -123 to calculate its $sinh$. The value is passed to comparison method and returns -1.3097586593E53 which is the correct value of $sinh(-123)$. Hence the test passes.
Test Case 6: $x = 12$	The test is run on double value 12 to calculate its $sinh$. The value is passed to comparison method and returns 81377.3957064 which is the correct value of $sinh(12)$. Hence the test passes.

Table 3: TestComparison Test Cases

2.2 Describe briefly how you went about testing and the computing environment that you used.

The testing is performed by using JUnit framework. JUnit 4 is a testing framework used for performing the tests. The JUnit 4 libraries are added to the JAR file. The test cases are run by Right clicking the java test file $->$ Run As $->$ JUnit Test. The test results are shown side by side of all the tests cases.The annotation @Test is used before the method to be tested. The test cases are run with pass or failure results. Below is the image of the test cases run on the function.



```
Finished after 0.065 seconds
Runs: 18/18    Errors: 0    Failures: 0

TestMyFuncs [Runner: JUnit 4] (0.001 s)
    testExponentialFun (0.001 s)
    testStrLengthFun (0.000 s)
TestComparison [Runner: JUnit 4] (0.000 s)
    testComparisonCase_A (0.000 s)
    testComparisonCase_B (0.000 s)
    testComparisonCase_C (0.000 s)
    testComparisonCase_D (0.000 s)
    testComparisonCase_E (0.000 s)
    testComparisonCase_F (0.000 s)
TestValidator [Runner: JUnit 4] (0.001 s)
    testAlphaFunCase_A (0.000 s)
    testAlphaFunCase_B (0.001 s)
    testSpaceFunCase_A (0.000 s)
    testSpaceFunCase_B (0.000 s)
    testMyValidatorFunCase_A (0.000 s)
    testMyValidatorFunCase_B (0.000 s)
    testMyValidatorFunCase_A1 (0.000 s)
    testSpecialCharFunCase_A (0.000 s)
    testSpecialCharFunCase_B (0.000 s)
    testNullFun (0.000 s)
```

```java
import org.junit.Test;
/**
 * This class contains test cases for the Class MyFuncs inside MyFuncs.java
 */
public class TestMyFuncs
{
    @Test
    public void testStrLengthFun()
    {
        Validator vad = new Validator();
        assertEquals(2, vad.strLen("12"));
    }

    @Test
    public void testExponentialFun()
    {
        Validator vad = new Validator();
        assertEquals(148.41310786,vad.exponential(20,5),1.0);
    }
}
```

```
<terminated> (default package) [JUnit] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (Aug 2, 2019, 1:33:36 AM)
Limit exceeds for value of x= Limit exceeds for value of x= The input should be numeric only.
The input contains spaces
The input should be numeric only. It cannot contain any alphabet.
The input contains spaces
The input contains special character #
```

Figure 13: Running Test Cases

3 References

- https://tex.stackexchange.com/
- https://www.freecodecamp.org/news/code-review-the-ultimate-guide-aa45c3
- https://en.wikipedia.org/wiki/PMD_(software)
- https://dzone.com/articles/java-code-review-checklist
- https://dzone.com/articles/junit-tutorial-setting-up-writing-and-runnin