SOEN 6011 - Software Engineering Processes
Deliverable 3

Submitted to
Prof. Pankaj Kamthan

By
Himansi Patel (40072262)
Github id : https://github.com/Himansipatel/SOEN-6011-Team-H-Himansi

Contents

0.1 Problem 5

0.1.1 Code review for function $f(x) = tan(x)$

Approach

A substantial portion of the code review has been done manually although PMD code review plugin was used at times to get a hands on experience on automated code review. PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth.[2]

Code review comment

As a team, we had a global agreement to use checkstyle as a plugin in IDE before beginning of development phase, so there were no issues regarding indentation, white space, missing comments, etc.
But here are possible improvements, with regards of Logic and Architecture style, which are listed below
then,

1. The below attached code part is leading to a redundancy factor as PI value is going to be same because actual parameters 100,99 is passed to a calling function which is fixed. So why not to follow a **singleton pattern**, which will calculate the value of PI if only required. In conclusion, instead of writing this code (Fig1) in every file, simply refactor the code by calling PIAve.getValue(); where getValue is a method which checks whether the PI value is already calculated, if yes then simply return else call the calcPI() method and return value.

```
9  +   static final double PI;
10 +
11 +   static {
12 +     PI = PIAve.calcPI(100, 99);
13 +   }
14 +
```

2. Avoid Hard Coding Value, create a constants file and register all constants value in that file.

```
8  +  * @author Nirav Patel
9  +  */
10 + public class PIAve {
11 +
12 +    public static DecimalFormat d16 = new DecimalFormat("0.0000000000000000");
```

Himansipatel 2 hours ago • edited

Avoid HardCoding Value, create a constants file and register all constants value in that file.
For example:
class Constant{
public static final String DECIMAL_FORAMT_VALUE = "0.0000000000000";
}

3. Proper naming convention to follow

```
15 +   static {
16 +     PI = PIAve.calcPI(100, 99);
17 +   }
18 +
19 +   static double tan(double i) {
```

Himansipatel 2 hours ago • edited

Missing JavaDoc comment and what is i?
A proper name could be given to convey what value the parameter is going to receive
for instance, it could have been an angle in degree or radian.

4. For the validator.java, it validates whether the input and output is in valid range or not, it checks for the different condition and prints the same message, which can be combined in single if condition by making use of logical OR condition.

```
14  +     * This method validates the range of the input.
15  +     * @param valueofInput input given by the user.
16  +     * @return return true if input is in range else return false
17  +     */
18  +    public static boolean affirmInputRange(double valueofInput) {
19  +      boolean result = true;
20  +      if (valueofInput > Double.MAX_VALUE) {
```

Himansipatel 2 hours ago

What is need of nested if-else here? It could be as:
if (valueofInput > Double.MAX_VALUE || valueofInput < -Double.MAX_VALUE) {
result = false;
}

Reply...

0.2 Problem 7

0.2.1 Test Cases Analysis for Function $f(x) = sinh(x)$

Functional testing is a phase in software development which involves testing and verifying the functional requirements of system. It is generally accomplished in various forms and levels like unit testing at developers end, integration testing and system testing by QA teams.
Its is a quality assurance (QA) process and is generally considered to be a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output.[1]
We have specifically used :
1. Black-box testing approach ignoring the internal design of the system and just ensuring if the results align well with the intended behaviour.
2. Some boundary value testing to handle edge cases that are not explicitly covered in the functional requirements.

Functional testing primarily involves the given steps

1. Identifying the core functionalities that the system is expected to perform
2. Creating/Mocking input data sets for the test cases, in alignment with system's functional requirements.
3. Determining outputs for the prepared input data sets.
4. Executing the test suit and thereon comparing actual and expected outputs.
5. Checking if the application behaves as intended and respect the functional specifications.

The summary tables for the test cases are described below which compares the actual output with the expected output based on input and give status.

Test Case Method : testInitCalculationInvalidInput()

This unit test is tested on improper inputs like string and combinations of special characters and it return the expected response.

Test Case Method : testInitCalculationInvalidUpperBoundInput()

This test task was to identify the upper bound value and if it exceeds the threshold value this unit function is returning the expected response.

Test Case Method : testInitCalculationValidInput()

This test is expected to receive the proper value as input and return the result, but this unit function works for some real integers.

Input	Expected Result	Actual Result	Status
9	4051.54190208	3991.013168667828000	Failed
1	1.1752011936438	1.1752011936438	Passed

Test Case Method : testEPowerXFinite()

This test case is expected to calculate Euler value with respective finite value of x and it is returning the expected value of e^x.

Test Case Method : testCalculateSinh()

This test is use for calculating value of sinh which accepts 2 parameters e^x and e^{-x} and returns the expected value for some real number of sinh function.

Input1	Input2	Expected Result	Actual Result	Status
9	-9	4051.54190208	3991.013168667828000	Failed
1	-1	1.1752011936438	1.1752011936438	Passed

Test Case Method : testSignificantDecimalPoints()

This test just verifies that input given by user, contains max 15 decimal significant points which will then use as processedInput.

Input	Expected Result	Actual Result	Status
0.000000000000000000000	0.000000000000000	0.000000000000000	Passed
0.12	0.12	0.12	Passed

Test Case Method : testValidateInputRangeInvalidInput()

This test is working correctly for detecting that the input value should lie within the given interval.

Input	Expected Result	Actual Result	Status
2*Double.MAX_VALUE	false	false	Passed

Test Case Method : testValidateOutputRange

This test is working properly for the cases where output reaches to too large value which cannot be stored or not in range.

Input	Expected Result	Actual Result	Status
2*Double.NEGATIVE_INFINITY	false	false	Passed
2*Double.POSITIVE_INFINITY	false	false	Passed
9	true	true	Passed

Results

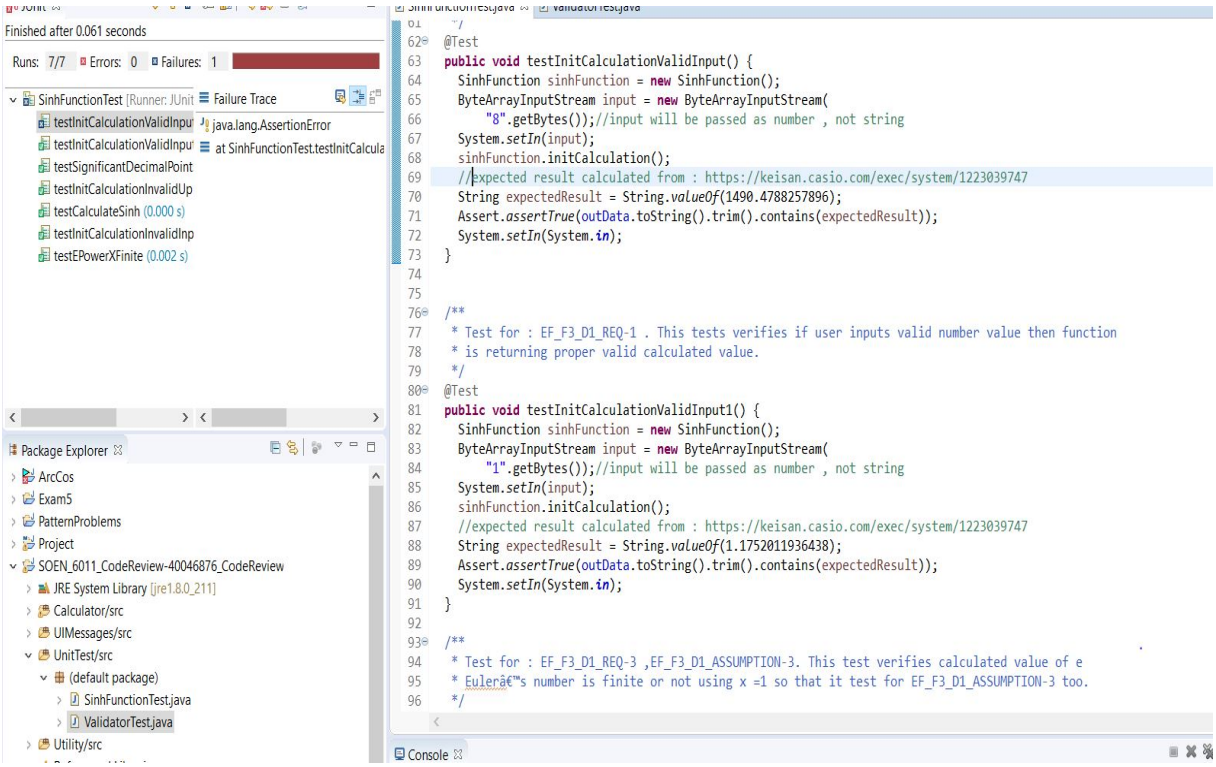

Figure 1: Execution of test cases for SinhFunctionTest.java

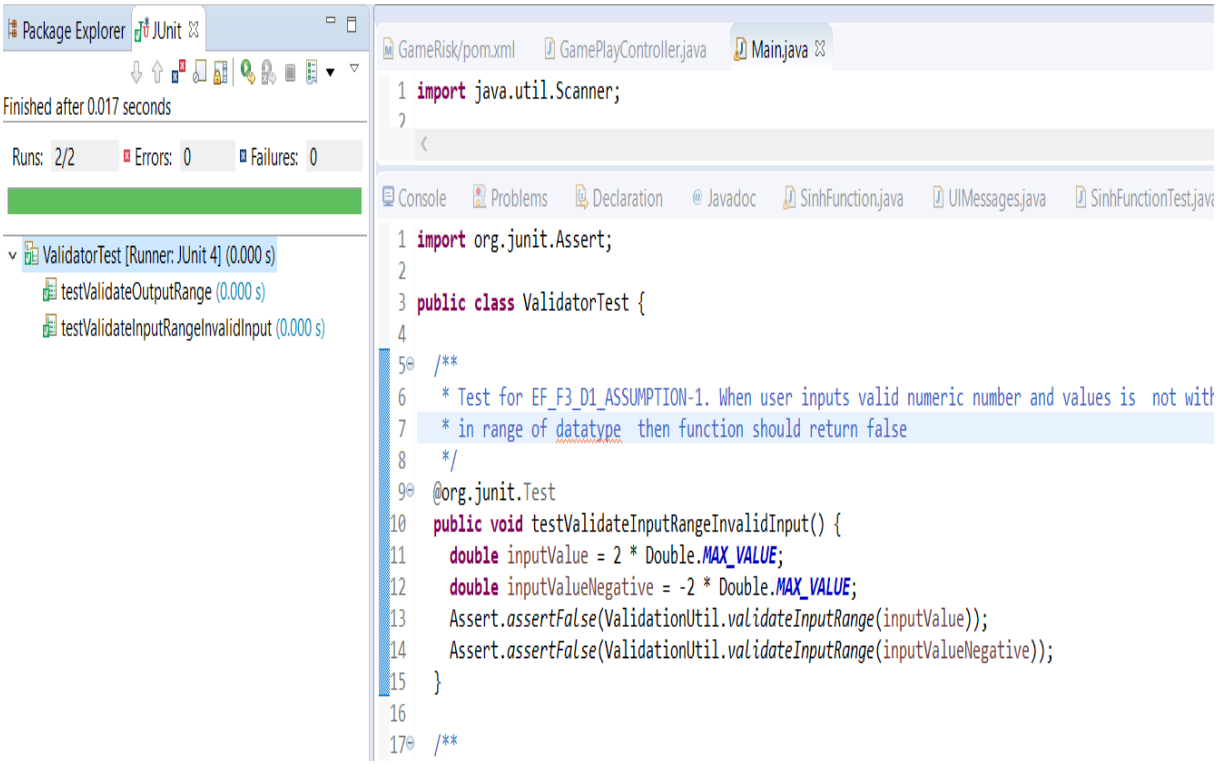

Figure 2: Execution of test cases for ValidatorTest.java

0.3 Version Control System

For code Review : -
https://github.com/niravjdn/SOEN-6011-Project-Team-H-Nirav/pull/2/files For Test Case Analysis : -
https://github.com/prashantp995/SOEN_6011_CodeReview

Bibliography

[1] https://www.softwaretestinghelp.com/guide-to-functional-testing/

[2] https://marketplace.eclipse.org/content/pmd-eclipse-plugin