SOEN 6011

$sinh(x)$

Problem - 4
Error Handling, Debugger and Quality Attributes

Author

Kolimi Dineshkumar Babu

Problem 4 - F3: $sinh(x)$

SOEN 6011 - Summer 2022 **Kolimi Dineshkumar Babu**
Software Engineering Processes **40094976**
Repository address : https://github.com/dineshkumarkolimi/Soen6011

Implementation Difficulties

There were mainly 2 difficulties to making the function implementation efficient and accurate.

Difficulty 1

The second problem was a binary floating point limitation of the double type. The integral result of the function with integer inputs was verified through testing to be accurate to at least 12 digits using the built-in Java $Math.sinh(x)$ function; however, the accuracy of decimal values varied in part due to the multiplication of the integral result by the decimal half. A $pm\ infty$ value was returned when e was increased to a value bigger than $pm709$ because doubles have a size limit.Due to an unsuccessful attempt to programme a new type that could handle math on extremely large decimal numbers with no floating point error and arbitrary precision, the cap for values of the integral and numerator component of the rationalised decimal remained $pm\ e(709)$.

Due to the aforementioned two issues, calculations beyond pm 709 returned $pm\ infty$ as is common for doubles, and there was a considerable reduction in accuracy for values with decimals that either could not reduce into fractions with small integers or were just too large to calculate the GCD rapidly.

Difficulty 2

The initial issue was the run-time of the root functions and largest common denominator. Any gains from reducing the size of the rational numbers were typically exceeded by the time needed to run the GCD function, which is roughly inversely proportional to the size of the input integers. The trend for the computation time of the root function increased with the root level.

Code Qualities

Despite the issues with decimal values mentioned above, integral number calculations are extremely accurate. Unit testing and manual testing quickly verified that the values were accurate. The subordinate functions are divided to promote reuse and modularity and maintainability, and the function code is correctly documented with in-depth Javadoc annotations. Magic numbers are defined as constants to allow for future programme changes. Each function's implementation can be called directly without the need to initialise an object because it is all stored within a single static library or utility class.

Error Handling

Issues with the input format are dealt with by the *sinh* function itself. In order to support numerous number formats and enhance accuracy over binary floating point inputs, it accepts a *String* as input. If it encounters unparsable data, it will throw a *NumberFormatException* class.

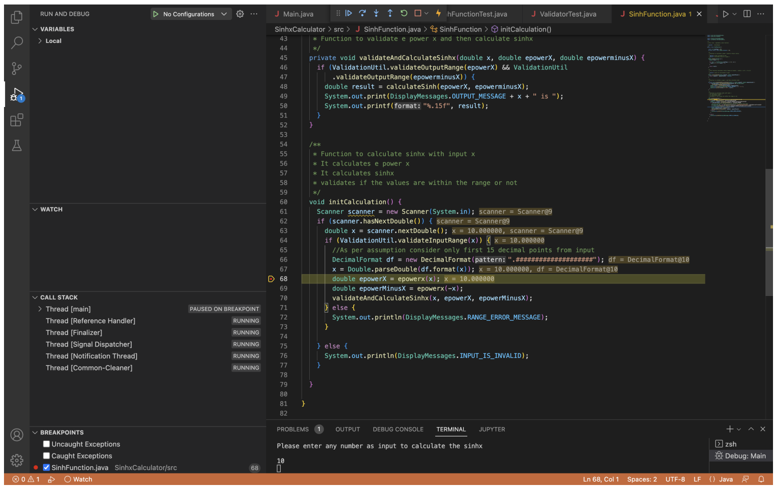
Checkstyle and Debugging

To verify the formatting of the source code, the Checkstyle tool was employed. Being a plug-in for the Eclipse IDE and having compatibility for both Google and Sun/Oracle styles made it simple to install and use. Although the code was mainly compliant with Checkstyle's rules, it did have a few irregularities. Since they are not modified in the function body, the majority of parameters and constants were afterwards adjusted to be final. Additionally, lines were divided to avoid using too many characters, and an else statement was moved to the line directly after the previous bracket. Given that it was a static utility/library class, the class was also given an empty private constructor to ensure that it could never be constructed.

There were just two issues that stuck out: it was difficult to find flaws like extra spaces on the lines where these "textit faults" were highlighted by Eclipse and/or Checkstyle, and style inconsistencies were labelled as "textit errors" rather than "textit warnings."

The snapshot of the checkstyle I used for this project is shown below.

Here are the screenshot of Debugger and Checkstyle which I used in this project respectively.



```
 * Function to validate e power x and then calculate sinhx
 */
private void validateAndCalculateSinhx(double x, double epowerX, double epowerminusX) {
  if (ValidationUtil.validateOutputRange(epowerX) && ValidationUtil
      .validateOutputRange(epowerminusX)) {
    double result = calculateSinh(epowerX, epowerminusX);
    System.out.print(DisplayMessages.OUTPUT_MESSAGE + x + " is ");
    System.out.printf(format: "%.15f", result);
  }
}

/**
 * Function to calculate sinhx with input x
 * It calculates e power x
 * It calculates sinhx
 * validates if the values are within the range or not
 */
void initCalculation() {
  Scanner scanner = new Scanner(System.in); scanner = Scanner@9
  if (scanner.hasNextDouble()) { scanner = Scanner@9
    double x = scanner.nextDouble(); x = 10.000000, scanner = Scanner@9
    if (ValidationUtil.validateInputRange(x)) { x = 10.000000
      //As per assumption consider only first 15 decimal points from input
      DecimalFormat df = new DecimalFormat(pattern: ".##################"); df = DecimalFormat@10
      x = Double.parseDouble(df.format(x)); x = 10.000000, df = DecimalFormat@10
      double epowerX = epowerx(x); x = 10.000000
      double epowerMinusX = epowerx(-x);
      validateAndCalculateSinhx(x, epowerX, epowerMinusX);
    } else {
      System.out.println(DisplayMessages.RANGE_ERROR_MESSAGE);
    }

  } else {
    System.out.println(DisplayMessages.INPUT_IS_INVALID);
  }

}

}
```

```
Thread [main]                      PAUSED ON BREAKPOINT
Thread [Reference Handler]         RUNNING
Thread [Finalizer]                 RUNNING
Thread [Signal Dispatcher]         RUNNING
Thread [Notification Thread]       RUNNING
Thread [Common-Cleaner]            RUNNING
```

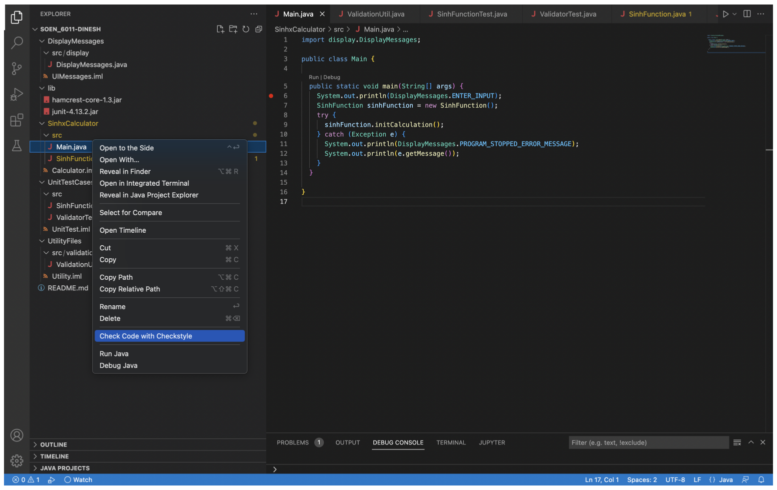
```
Uncaught Exceptions
Caught Exceptions
SinhFunction.java  SinhxCalculator/src  68
```

```
Please enter any number as input to calculate the sinhx

10
```

```
0 1    Ln 68, Col 1  Spaces: 2  UTF-8  LF  {} Java
```

Figure 1: Debugger screenshot



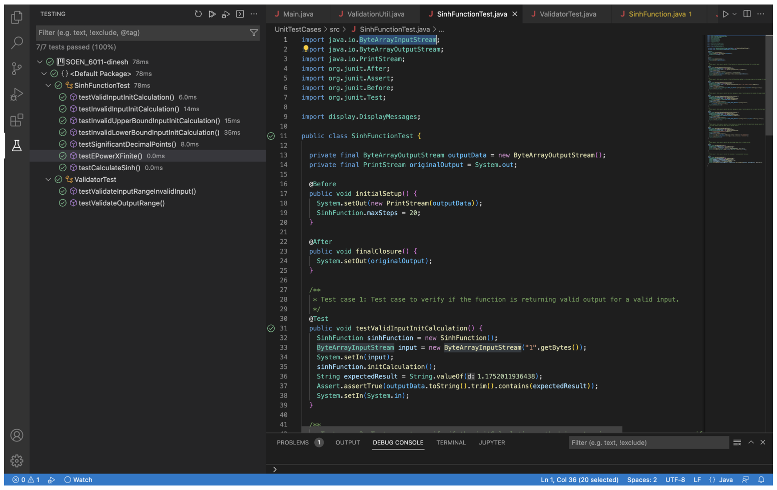
```java
import display.DisplayMessages;

public class Main {

  public static void main(String[] args) {
    System.out.println(DisplayMessages.ENTER_INPUT);
    SinhFunction sinhFunction = new SinhFunction();
    try {
      sinhFunction.initCalculation();
    } catch (Exception e) {
      System.out.println(DisplayMessages.PROGRAM_STOPPED_ERROR_MESSAGE);
      System.out.println(e.getMessage());
    }
  }

}
```

Figure 2: Checkstyle screenshot



```
7/7 tests passed (100%)
SOEN_6011-dinesh 78ms
  <Default Package> 78ms
    SinhFunctionTest 78ms
      testValidInputInitCalculation() 6.0ms
      testInvalidInputInitCalculation() 14ms
      testInvalidUpperBoundInputInitCalculation() 15ms
      testInvalidLowerBoundInputInitCalculation() 35ms
      testSignificantDecimalPoints() 8.0ms
      testEPowerXFinite() 0.0ms
      testCalculateSinh() 0.0ms
    ValidatorTest
      testValidateInputRangeInvalidInput()
      testValidateOutputRange()
```

```java
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import display.DisplayMessages;

public class SinhFunctionTest {

  private final ByteArrayOutputStream outputData = new ByteArrayOutputStream();
  private final PrintStream originalOutput = System.out;

  @Before
  public void initialSetup() {
    System.setOut(new PrintStream(outputData));
    SinhFunction.maxSteps = 20;
  }

  @After
  public void finalClosure() {
    System.setOut(originalOutput);
  }

  /**
   * Test case 1: Test case to verify if the function is returning valid output for a valid input.
   */
  @Test
  public void testValidInputInitCalculation() {
    SinhFunction sinhFunction = new SinhFunction();
    ByteArrayInputStream input = new ByteArrayInputStream("1".getBytes());
    System.setIn(input);
    sinhFunction.initCalculation();
    String expectedResult = String.valueOf(d: 1.1752011936438);
    Assert.assertTrue(outputData.toString().trim().contains(expectedResult));
    System.setIn(System.in);
  }

  /**
```

Figure 3: Testcases screenshot