# SOEN 6011
# Software Engineering Processes
# Project Report
# F3: sinh(x)
# Supervised by:
# Dr. Pankaj Kamthan

Sai Sankeerth Koduri (Id: 40195685)

5th August,2022

## 1 Problem 1

### 1.1 Brief Description

Sinh(x) is short for Hyperbolic sine of the element x. The hyperbolic sine (and cosine) is a linear combination of two exponents of the Euler number, e. In mathematics, hyperbolic functions are analogues of the ordinary trigonometric functions, but defined for the unit hyperbola rather than on the unit circle.[2]

Hyperbolic functions occur in the calculations of angles and distances in hyperbolic geometry. They also occur in the solutions of many linear differential equations, cubic equations, and Laplace's equation in Cartesian coordinates.

Sinh(x) is defined as, **sinh(*x*)** = ($e^x$ - $e^{-x}$)÷ 2, Where e is the Euler's number (base for natural logarithms).

### 1.2 Domain and Co-Domain

The domain and co-domain of sinh(x) is the set of Real numbers, $\mathbb{R}$ [3]

### 1.3 Characteristics that make it unique

- Sinh(x) along with cosh x gives us all the points on the unit hyperbola i.e, $x^2 - y^2 = 1$ which in-turn gives rise to lot of hyperbolic trigonometric identities which can be used for parametrizing and solving integrals.[4]
- Tanh(x) defined as sinh(x)/cosh(x) describes the geometry of Special Theory of Relativity.[5]

- The properties of the catenary are nicely described using the hyperbolic trigonometric functions. A catenary is the curve you get by hanging a chain of uniform density by its two endpoints.
- item sinh($x$) $\approx$ cosh ($x$) for large x. Similarly, sinh($x$) $\approx$ -cosh ($x$) for large negative x.
- sinh($x$) is odd function , sinh($-x$) = - sinh($x$) and it has a period of $2\pi\imath$

### 1.4 Context of Use Model

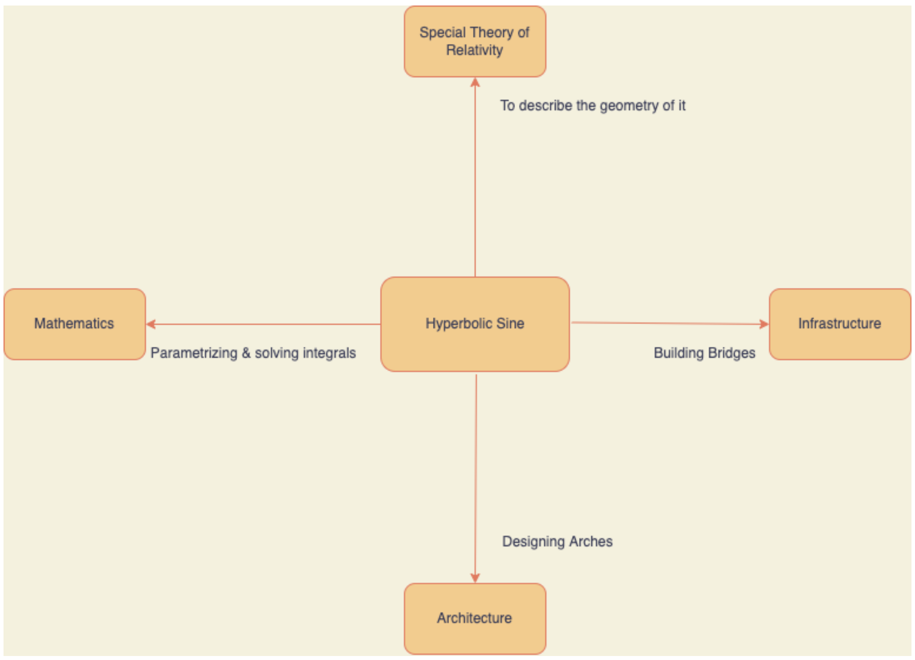

Special Theory of Relativity
To describe the geometry of it
Mathematics
Parametrizing & solving integrals
Hyperbolic Sine
Infrastructure
Building Bridges
Designing Arches
Architecture

Figure 1: Context of use Model

## 2 Problem 2

### 2.1 Requirements

1. Requirement 1
**ID: ET-F3-Sinh-REQ_1**
Type : Functional Requirement
Description: The function should return a hyperbolic sine of input, if the user provides a real number as an input.

2. Requirement 2
   **ID: ET-F3-Sinh-REQ_2**
   Type : Functional Requirement
   Description: The function should throw an error message stating that it is an invalid input, if the user provides an alphabet or a special character as an input.

3. Requirement 3
   **ID: ET-F3-Sinh-REQ_3**
   Type : Functional Requirement
   Description: The calculated output value should be finite. If the value is not finite, it should throw an error message stating the output is not finite.

4. Requirement 4
   **ID: ET-F3-Sinh-REQ_4**
   Type : Non Functional Requirement
   Description: The project should be designed in a way that it is modular and not tightly coupled.

## 2.2 Assumptions

1. Assumption 1
   **ID: ET-F3-Sinh-AS_1**
   Description: Even though the hyperbolic sine function has a domain of all possible real numbers, as the function will be implemented using java programming language, there is a limitation on user input because of the datatype.
   The maximum input is + 1.79769313486231570E+308 and minimum input is - 1.79769313486231570E+308.
2. Assumption 2
   **ID: ET-F3-Sinh-AS_2**
   Description: Even if user inputs maximum allowed value,the output will be limited and output will be less than maximum allowed value and more than minimum allowed value.
   The maximum input is + 1.79769313486231570E+308 and minimum input is - 1.79769313486231570E+308.
3. Assumption 3
   **ID: ET-F3-Sinh-AS_3**
   Description: First 15 decimal points will be considered as a significant decimal point when the user will provide an input.
4. Assumption 4
   **ID: ET-F3-Sinh-AS_4**
   The value of e raised to x is calculated using the Taylor series,which goes upto infinity.Since there is limitation of datatype in java programming language infinite calculation is not possible.This is why we stop after the first 20 steps.

## 3 Problem 3

#### 3.0.1 Algorithm 1

**Description:** In this algorithm, the value of e(Euler′s number) is not calculated explicitly. The value of e is declared as a constant and its value is copied from the e value in java.lang.Math library.
This constant is then used to calculate the values of $e^{x}$ and $e^{-x}$ which are further used to obtain sinh(x).
**Advantage:**

- No explicit method call is required to calculate the value of *e*.

**Disadvantages:**

- No way of increasing the accuracy of e as it is declared as a constant.
- Possibility of corruption of the value of e.

**Reasons to select this algorithm:**

- No Recursion.
- Simple to implement.

**Pseudocode for Algorithm 1:**

```
Algorithm 1 sinh($x$) = ($e^x - e^{-x}$)$\div$ 2
Require: $x \neq null$
Ensure: $x \neq null$ and sinh($x$) = ($e^x - e^{-x}$)$\div$ 2
  $e$ ←Declared as a constant
  $e^x$ ←GETERAISEDTOX($x$)
  $e^{-x}$ ←GETERAISEDTOX($-x$)
  $sinhx \leftarrow (e^x - e^{-x}) \div 2$

  function GETERAISEDTOX($x$)
  $result \leftarrow e$

  if $x$ >0 then
     while $counter < x$ do
        $result = result * e$
        $counter$ ←counter+1
     end for
  end while
  if $x$ <0 then
        $y$ ←-x
        while $counter < y$ do
        $result = result * e$
        $counter$ ←counter+1
     end while
  end if
  if $x$ =0 then
     $result = 1$
  end if
     return $result$
  end function=0
```

### 3.0.2 Algorithm 2

**Description:**
In this algorithm,value of e is not calculated explicitly.The method GETERAISEDTOX calculates value of e raised to x using the Taylor series.[6] Since the algorithm should terminate at some point , as we can not calculate the Taylor series till infinity, we terminate the method when the counter value reaches 20.This gives us a very decent accuracy while calculating sinh(x).

**Advantages:**

- Accuracy can be increased by increasing the counter value where the function terminates
- e is not declared as a constant and there is no chance in corruption of the value which is likely in Algorithm 1.

**Disadvantages:**

- Deciding the counter value where the method should terminate is difficult.

**Reasons to select this algorithm:**

- Flexibility in achieving an accurate result for sinh($x$) by increasing the the counter value at which the method will terminate.
- No chance in corruption of the value of e which is possible in Algorithm 1.
- Need no know the value of e beforehand, as we do not store it as a constant

**Pseudocode for Algorithm 2:**

**Algorithm 2** Calculate sinh($x$) = ($e^{x} - e^{-x}$)$\div$ 2

```
Require: $x \neq null$
Ensure: $x \neq null$ and sinh($x$) = ($e^{x} - e^{-x}$)$\div$ 2
  $e^{x} \leftarrow$GETERAISEDTOX($x$)
  $e^{-x} \leftarrow$GETERAISEDTOX($-x$)
  $sinhx \leftarrow (e^{x} - e^{-x}) \div 2$

  function GETERAISEDTOX($x$)
  $result \leftarrow 1$
  $counter \leftarrow 20$
  while $counter > 0$ do
    $result = 1 + ($x$*$result$) \div counter$;
    counter $\leftarrow counter$-1
  end while
    return $result$
  end function=0
```

Considering the advantages and disadvantages of both the algorithms, Algorithm 2 is implemented.

### 3.1 Mind map

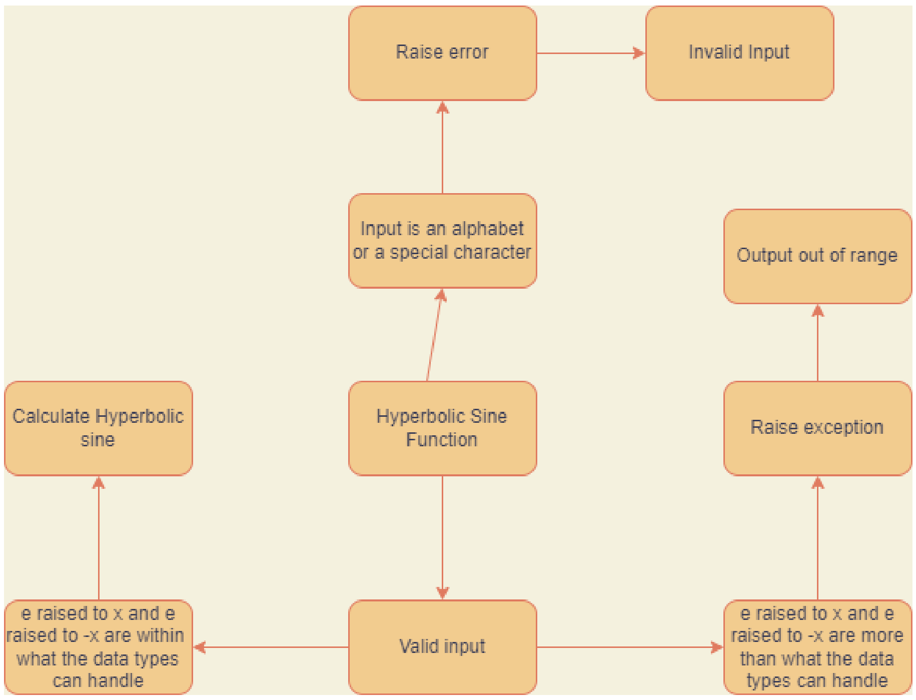

Raise error
Invalid Input
Input is an alphabet or a special character
Output out of range
Calculate Hyperbolic sine
Hyperbolic Sine Function
Raise exception
e raised to x and e raised to -x are within what the data types can handle
Valid input
e raised to x and e raised to -x are more than what the data types can handle

Figure 2: Mind map for selecting the pseudo code format

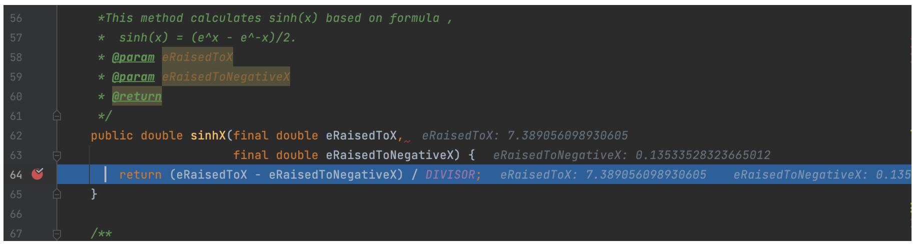
## 4 Problem 4

### 4.1 Debugger

A debugger is a computer program used by programmers to debug and test their programs. While programming, debugger helps to detect and resolve any bugs or unexpected behaviour.[7] During implementation of F3: sinh($x$) using java, I used IntelliJ debugger provided by the IntelliJ IDE.

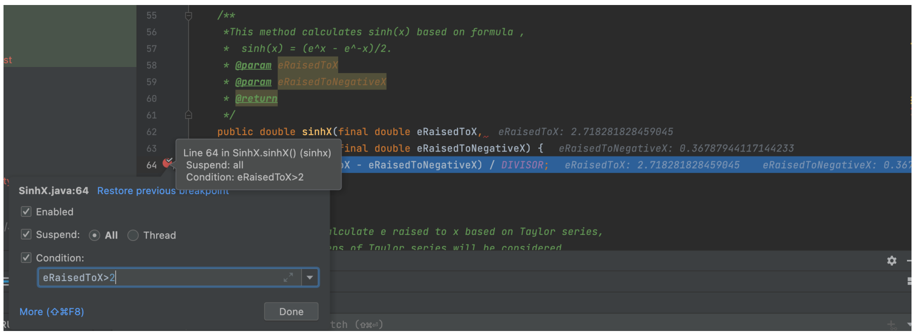
**Advantages of IntelliJ Debugger**

- IntelliJ debugger provides for an inline view while debugging.
- supports conditional breakpoints.
- Create breakpoints and watchpoints and also configure them.
- Pause and resume the debugger session at any time.
- Can debug Java applications remotely.
- shows the memory usage which can be used for analysis.



```
56      *This method calculates sinh(x) based on formula ,
57      *  sinh(x) = (e^x - e^-x)/2.
58      * @param eRaisedToX
59      * @param eRaisedToNegativeX
60      * @return
61      */
62     public double sinhX(final double eRaisedToX,   eRaisedToX: 7.389056098930605
63                         final double eRaisedToNegativeX) {   eRaisedToNegativeX: 0.13533528323665012
64         return (eRaisedToX - eRaisedToNegativeX) / DIVISOR;   eRaisedToX: 7.389056098930605    eRaisedToNegativeX: 0.135
65     }
66
67     /**
```

Figure 3: Inline value view in IntelliJ debugger(Values in gray font)



```
55     /**
56      *This method calculates sinh(x) based on formula ,
57      *  sinh(x) = (e^x - e^-x)/2.
58      * @param eRaisedToX
59      * @param eRaisedToNegativeX
60      * @return
61      */
62     public double sinhX(final double eRaisedToX,   eRaisedToX: 2.718281828459045
63                         final double eRaisedToNegativeX) {   eRaisedToNegativeX: 0.36787944117144233
64         X - eRaisedToNegativeX) / DIVISOR;   eRaisedToX: 2.718281828459045    eRaisedToNegativeX: 0.367
```

Line 64 in SinhX.sinhX() (sinhx)
Suspend: all
Condition: eRaisedToX>2

SinhX.java:64 Restore previous breakpoint

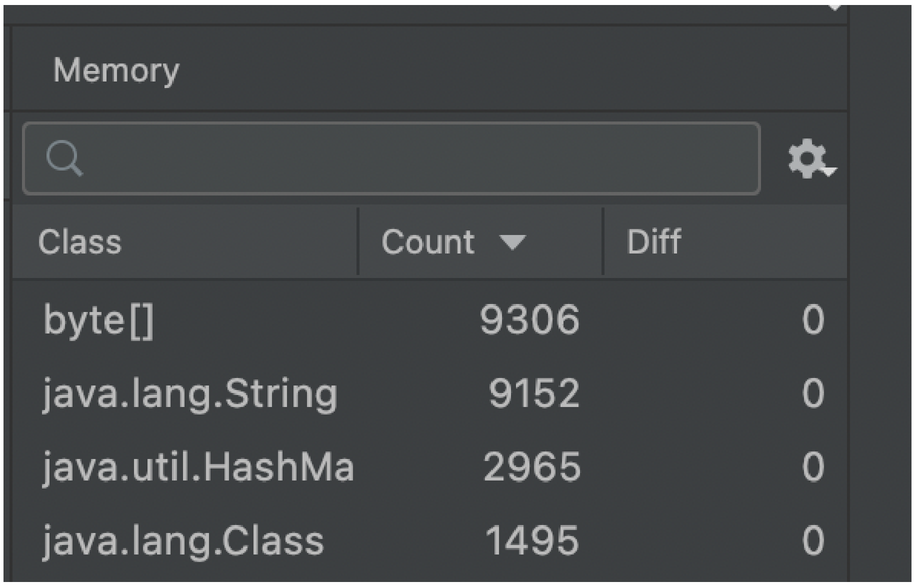
[✓] Enabled

[✓] Suspend: [✓ All] Thread

[✓] Condition:

eRaisedToX>2

Figure 4: Support for conditional break points



Memory

| Class | Count | Diff |
|---|---|---|
| byte[] | 9306 | 0 |
| java.lang.String | 9152 | 0 |
| java.util.HashMa | 2965 | 0 |
| java.lang.Class | 1495 | 0 |

Figure 5: Support for checking memory

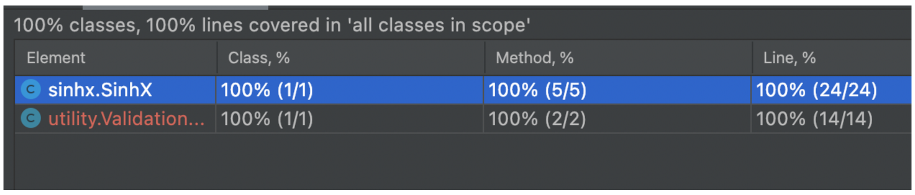
**Disadvantages of IntelliJ Debugger**

- It is not possible to use IntelliJ Debugger independently.
- IntelliJ debugger requires high end configurations and consumes more memory compared to the other debuggers.

## 4.2 Quality Attributes

1. **Correctness**

    - The correctness of the program is checked using the unit test cases.[11]
    - With the help of these test cases, the expected value and the actual value are compared.
    - Code coverage of the test cases is 100% to ensure the correctness and avoid unexpected behaviour



100% classes, 100% lines covered in 'all classes in scope'

| Element | Class, % | Method, % | Line, % |
|---|---|---|---|
| sinhx.SinhX | 100% (1/1) | 100% (5/5) | 100% (24/24) |
| utility.Validation... | 100% (1/1) | 100% (2/2) | 100% (14/14) |

Figure 6: Code coverage for the Project

2. **Efficiency**

    - No nested loops used.
    - No recursion used.
    - Junit test suit (8 test cases) which executes all implemented functions takes 53 ms to complete.

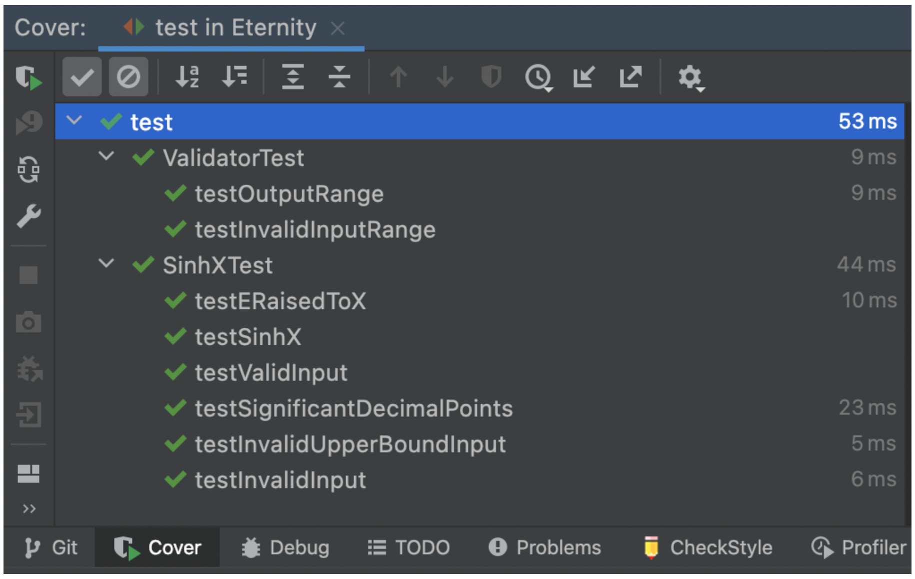

```
Cover:  test in Eternity
test                                   53 ms
  ValidatorTest                         9 ms
    testOutputRange                     9 ms
    testInvalidInputRange
  SinhXTest                            44 ms
    testERaisedToX                     10 ms
    testSinhX
    testValidInput
    testSignificantDecimalPoints       23 ms
    testInvalidUpperBoundInput          5 ms
    testInvalidInput                    6 ms
```

Figure 7: Junit test suit Execution time for all functions

3. **Robustness**
   The implementation is robust because:
   - Strict validation is carried out on the user input.
   - checking the input to see if it is compatible with the data types before any calculations.
   - If the output is more than the range supported by the data types, an error message is thrown.
4. **Usability**
   Usability is achieved because:
   - No complicated ui is used. A user-friendly command line ui is being used.
   - Error messages are informative and to the point.
   - Asks the users if want to continue or terminate the application.
   - The project can be exported as executable jar file.
5. **Maintainable**
   - Maintainability is being achieved using a modular design.
   - Informative comments are added.

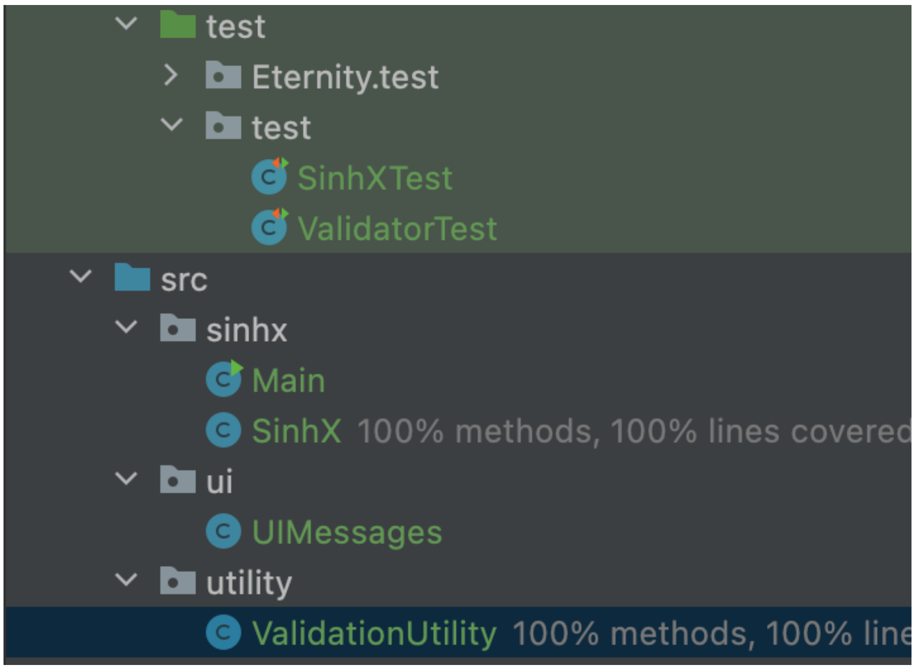

test
Eternity.test
test
SinhXTest
ValidatorTest
src
sinhx
Main
SinhX 100% methods, 100% lines covered
ui
UIMessages
utility
ValidationUtility 100% methods, 100% line

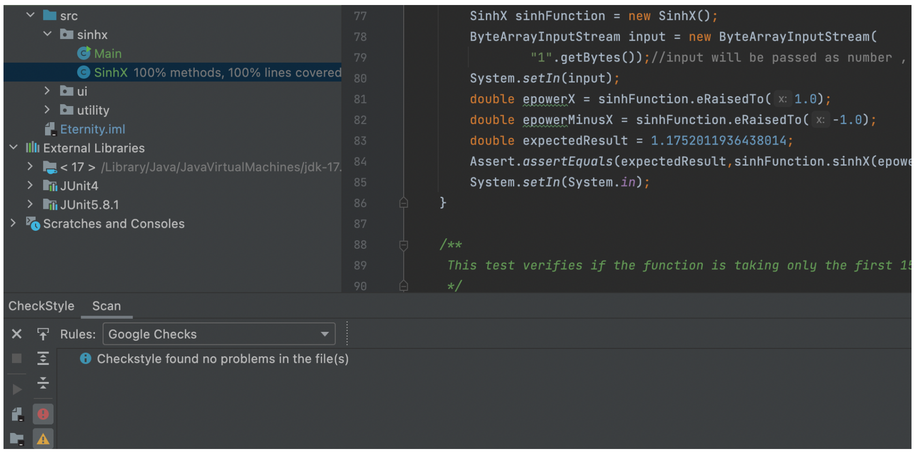
Figure 8: Project Structure showing the modular design

6. **Error Message and Exception Handling**

   - Responding to unexpected or undesirable events that occur while a computer program is running is known as exception handling. Without this procedure, exceptions would disrupt a program's normal operation and lead to a crash. To avoid this, exception handling is used.
   - In the Hyperbolic sine Function Calculator, if a user enters any other input other than real numbers(like alphabets or special characters), then the user will be notified by an error message ”Please enter a valid input” Moreover, if the user enters a number which is more than a java datatype can handle, it notifies with another error message” and for any general exception a message ”Fatal error” is thrown by an exception handling mechanism

## 5 CheckStyle

CheckStyle is a tool which helps programmers write Java code that adheres to a coding standard.
Google CheckStyle[10] is used in this project



```
        SinhX sinhFunction = new SinhX();
        ByteArrayInputStream input = new ByteArrayInputStream(
                "1".getBytes());//input will be passed as number ,
        System.setIn(input);
        double epowerX = sinhFunction.eRaisedTo( x: 1.0);
        double epowerMinusX = sinhFunction.eRaisedTo( x: -1.0);
        double expectedResult = 1.1752011936438014;
        Assert.assertEquals(expectedResult,sinhFunction.sinhX(epowe
        System.setIn(System.in);
    }

    /**
     This test verifies if the function is taking only the first 15
     */
```

Rules: Google Checks

Checkstyle found no problems in the file(s)

Figure 9: Google CheckStyle being used

### Advantages

- Easy to configure. It can be integrated into the build such that the project will build only if there are no checkStyle errors.
- Makes the code more readable.

### Disadvantages

- No check that finds redundant type casts.
- Better alternatives such as sonarQube are available in the market.

### 5.1 Programming style

Style guides ensure consistency and predictability across the project[8]. I am using Google Programming style for java. Reasons to use Google Style guide :

- It makes code easier to read.
- It allows programmer to focus on logic instead of making style decision
- It makes file and variable name more predictable.
- Programmer can import the style guide in IDE and reformat code with the help of short-cuts.

## 6 Problem 5

#### 6.0.1 Junit Standards

With the help of increased presentation and structuring of the usage of JUnit and testing, one can increase understanding and appreciation of the overall value of testing in software development.[9]

### 6.1 Junit traceability

### 6.2 Traceability to Requirements

1. **ET-F3-Sinh-REQ_1**
   Junit test case(s) : testValidInput(), testSinhX()
   Description : These tests verify if user provides a valid input, the function is showing a correct valid value.
2. **ET-F3-Sinh-REQ_2**
   Junit test case(s) : testInvalidInput()
   Description : This test verifies if user inputs invalid value (alphabet/special characters).If so, the function throws a proper error message or not
3. **ET-F3-Sinh-REQ_3**
   Junit test case(s) : testERaisedToX()
   Description : This test verifies obtained value of $e^x$ (Euler′s number) is finite or not.

## References

[1] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[2] https://en.wikipedia.org/wiki/Hyperbolic_functions

[3] https://www.analyzemath.com/DomainRange/domain_range_functions.html.

[4] https://www.quora.com/What-is-sinh-x

[5] https://math.stackexchange.com/questions/1070254/what-is-the-importance-of-sinhx

[6] https://en.wikipedia.org/wiki/Taylor_series

[7] https://raygun.com/blog/java-debugging-tool

[8] https://www.codereadability.com/why-use-a-style-guide

[9] MichaelWick,DanielE.Stevenson,andPaulJ.Wagner.Usingtestingandjunitacrossthecurriculum, 022005.

[10] https://checkstyle.sourceforge.io/writingchecks.html.

[11] https://en.wikipedia.org/wiki/Debugger

[12] https://www.modernrequirements.com/blogs/how-to-make-your-requirement-ids-more-descriptive-an

[13] https://standards.ieee.org/ieee/29148/6937/#:~:text=29148%2D2011,-ISO%2FIEC%2FIEEE&text=ISO%2FIEC%2FIEEE%2029148%3A,and%20their%20content%20are%20defined.