# DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: MOLDOVANU TUDOR
GROUP: 30421

## CONTENTS

1. Assignment Objective ........ 3
2. Problem Analysis, Modeling, Scenarios, Use Cases ........ 3
3. Design ........ 5
4. Implementation ........ 8
5. Results ........ 13
6. Conclusions ........ 14
7. Bibliography ........ 14

## 1. Assignment Objective

For Assignment 1, the primary objective is to implement a user application where users can input polynomials and perform various operations on them. These operations include addition, subtraction, multiplication, division, finding the derivative of a polynomial and computing the integral form of a polynomial. To achieve this, it is necessary to complete a couple of sub-objectives to ensure that the final product is functional and meets all requirements.

### SUB-OBJECTIVES:

1.Establishing the inputs/outputs of the application (aspect considered in the Problem Analysis Chapter);

2.The capabilities of the calculator regarding the operations performed (an aspect considered in the Problem Analysis chapter);

3.User interaction within the final product (aspect considered in the Problem Analysis Chapter and Implementation Chapter);

4.The implementation of the operations (aspect considered in the Implementation Chapter);

5.The GUI (Graphical User Interface) layout (aspect considered in the Implementation Chapter);

6.The structure and architecture of the application (aspect considered in the Design Chapter);

7.The interplay between packages and their contents (aspect considered in the Design Chapter);

8.Establishing the representation for the polynomials (aspect considered in the Design chapter).

9.Testing the code (aspect considered in the Results Chapter).

## 2. Problem Analysis, Modeling, Scenarios, Use Cases

For a correct and comprehensive implementation, the application should support various operations using polynomials of a single variable with integer coefficients as input. The results of the operations must be displayed on the screen for the user to easily interpret and utilize.

## FUNCTIONAL REQUIREMENTS:

1.It should provide options for users to select the desired mathematical operation, including addition, subtraction, multiplication, division, integration, and differentiation.

2.The calculator should accurately compute the result of the selected operation on the input polynomials.

3.The calculator should have a "clear" or "reset" function to allow users to start a new calculation easily.

4.The polynomial calculator should present the outcome of the operation on the screen for user viewing.

## NON-FUNCTIONAL REQUIREMENTS:

1.The polynomial calculator's user interface should have a clean and intuitive design for ease of use.

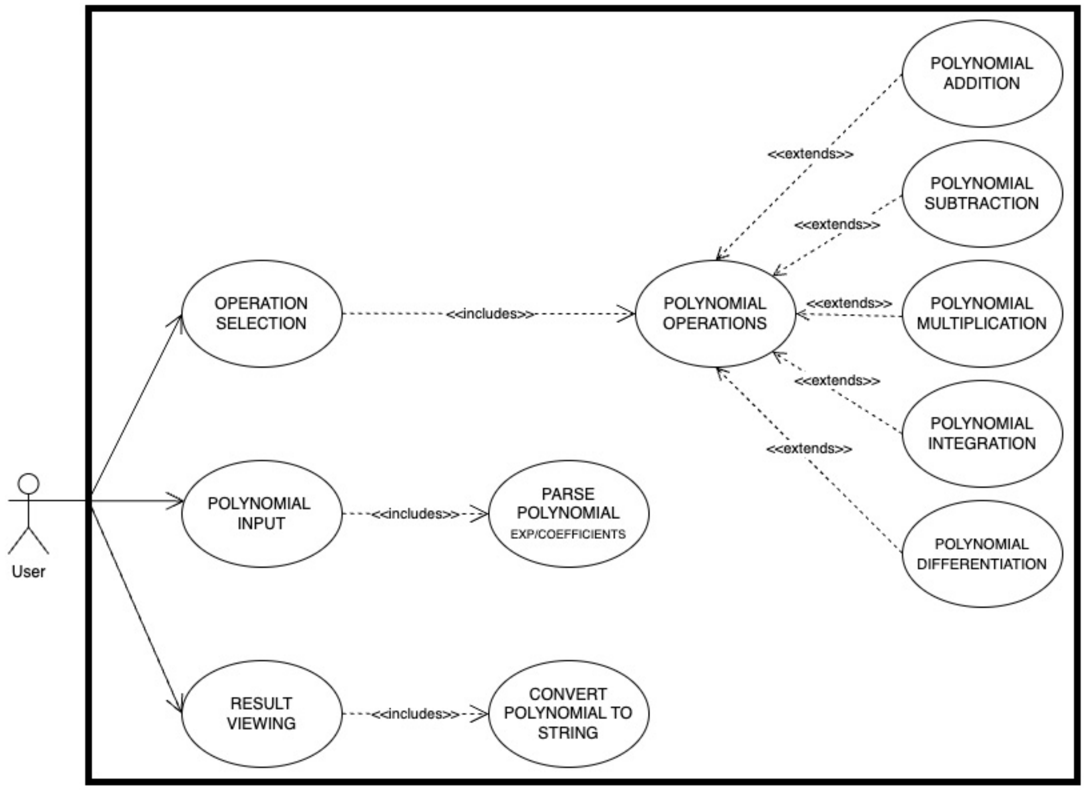
2.The application should have efficient error handling to provide clear error messages to users.

3.It should have a responsive design, ensuring smooth operation on various screen sizes and devices.

4.The application should be platform-independent, running smoothly on different operating systems.

5.The calculator should have fast computation times, especially for complex polynomials.

USE CASE DIAGRAM:



# 3. Design

## PACKAGES AND UML PACKAGE DIAGRAM

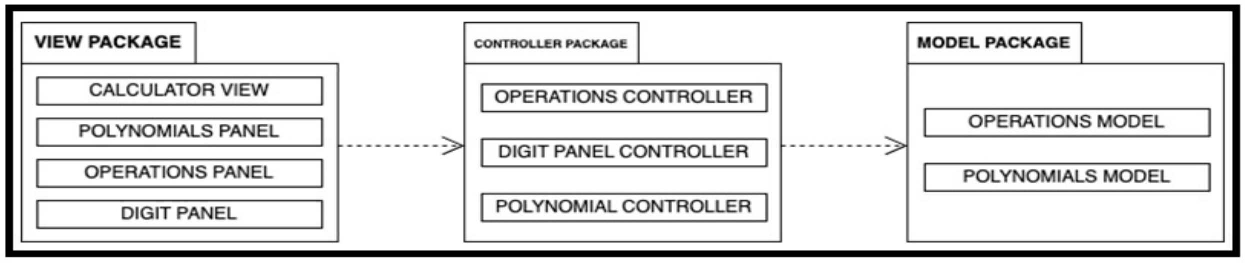
For this project's purposes, the most suitable architecture is the three-layered MVC (Model-View-Controller) architecture.

## PACKAGES

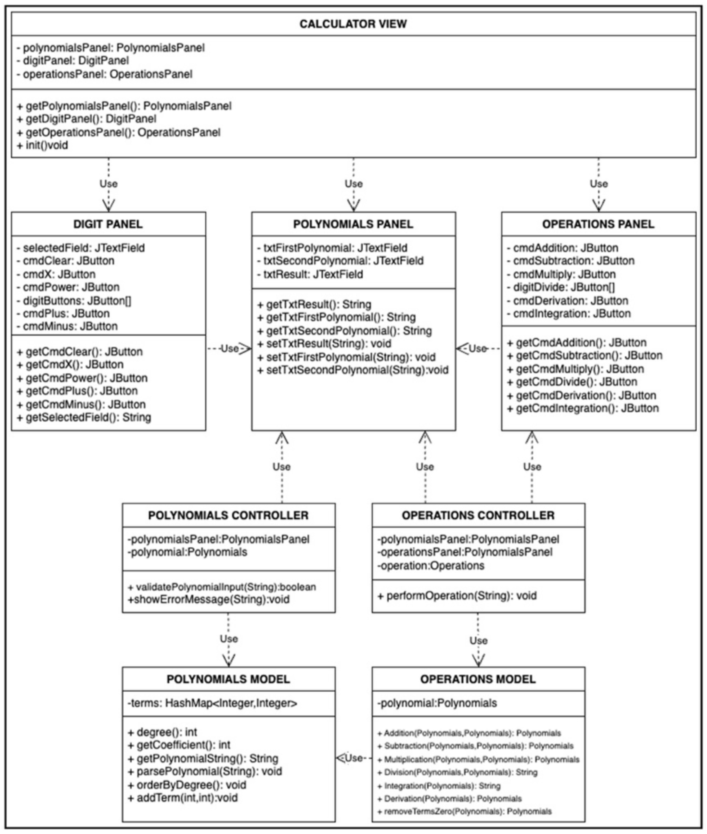
The View Package comprises GUI panels for the User Interface, including the Calculator View, Digit Panel, Polynomials Panel, and Operations Panel. These are distinct classes designed to enhance the overall design, featuring buttons (in the Digit and Operations Panels) and text fields (in the Polynomials Panel) as integral components for a smooth and user-friendly interface. The connection between the View Package and the Controller Package is established through the use of action listeners. This enables the Controller Package to manipulate the data transmitted on the

User Interface, facilitating the development of the application's backend tasks such as input validation and calling operation methods. Once the input is formatted in the Controller Package, the correct input is transmitted into the Model Package. Here, utilizing the data processed by the controller, the output of the operations is generated. The results of the operations are then sent back to the Controller Package for formatting, ready to be displayed on the Result Text Field within the View.

## UML PACKAGE DIAGRAM



## CLASS DESIGN AND UML CLASS DIAGRAM


CALCULATOR VIEW
- polynomialsPanel: PolynomialsPanel
- digitPanel: DigitPanel
- operationsPanel: OperationsPanel
+ getPolynomialsPanel(): PolynomialsPanel
+ getDigitPanel(): DigitPanel
+ getOperationsPanel(): OperationsPanel
+ init()void
Use
Use
Use
DIGIT PANEL
- selectedField: JTextField
- cmdClear: JButton
- cmdX: JButton
- cmdPower: JButton
- digitButtons: JButton[]
- cmdPlus: JButton
- cmdMinus: JButton
+ getCmdClear(): JButton
+ getCmdX(): JButton
+ getCmdPower(): JButton
+ getCmdPlus(): JButton
+ getCmdMinus(): JButton
+ getSelectedField(): String
Use
POLYNOMIALS PANEL
- txtFirstPolynomial: JTextField
- txtSecondPolynomial: JTextField
- txtResult: JTextField
+ getTxtResult(): String
+ getTxtFirstPolynomial(): String
+ getTxtSecondPolynomial(): String
+ setTxtResult(String): void
+ setTxtFirstPolynomial(String): void
+ setTxtSecondPolynomial(String):void
Use
OPERATIONS PANEL
- cmdAddition: JButton
- cmdSubtraction: JButton
- cmdMultiply: JButton
- digitDivide: JButton[]
- cmdDerivation: JButton
- cmdIntegration: JButton
+ getCmdAddition(): JButton
+ getCmdSubtraction(): JButton
+ getCmdMultiply(): JButton
+ getCmdDivide(): JButton
+ getCmdDerivation(): JButton
+ getCmdIntegration(): JButton
Use
Use
Use
POLYNOMIALS CONTROLLER
-polynomialsPanel:PolynomialsPanel
-polynomial:Polynomials
+ validatePolynomialInput(String):boolean
+showErrorMessage(String):void
OPERATIONS CONTROLLER
-polynomialsPanel:PolynomialsPanel
-operationsPanel:PolynomialsPanel
-operation:Operations
+ performOperation(String): void
Use
Use
POLYNOMIALS MODEL
-terms: HashMap<Integer,Integer>
+ degree(): int
+ getCoefficient(): int
+ getPolynomialString(): String
+ parsePolynomial(String): void
+ orderByDegree(): void
+ addTerm(int,int):void
Use
OPERATIONS MODEL
-polynomial:Polynomials
+ Addition(Polynomials,Polynomials): Polynomials
+ Subtraction(Polynomials,Polynomials): Polynomials
+ Multiplication(Polynomials,Polynomials): Polynomials
+ Division(Polynomials,Polynomials): String
+ Integration(Polynomials): String
+ Derivation(Polynomials): Polynomials
+ removeTermsZero(Polynomials): Polynomials

The primary UI component is established within the Calculator View class, which holds references to the three design elements: the Digit Keyboard Panel, the Operations Panel, and the Polynomial Text Fields Panel. The Calculator View serves as the user-program interface, through which the user can interact with the program.

The core logic is encapsulated within the Controller Package, consisting of two separate classes: Polynomials Controller and Operations Controller.

The Polynomials Controller manages the Polynomial Panel, verifying user-inputted text using Regex. If the input message is incorrect, an error message is displayed on the screen.

The Operations Controller manages the buttons within the Operation Panel, determining which operation to execute.

The Polynomials Model is the class that handles the modeling of polynomials by storing them in a HashMap (HashMap<Integer, Integer>, where the key represents the exponent and the value represents the coefficient). A Hash Map implementation is used because it offers fast access and retrieval of polynomial terms based on their exponents. This is advantageous for a polynomial calculator because:

1. Fast Access: Hash maps provide constant-time (O(1)) access to elements, meaning that retrieving or updating polynomial terms based on their exponents is very efficient, especially for large polynomials.

2. Simplified Operations: With the exponent as the key and coefficient as the value, performing operations such as addition, subtraction, multiplication, and division becomes straightforward. The hash map structure allows for easy manipulation of polynomial terms.

3. Flexibility: Hash maps allow for dynamic resizing, meaning the polynomial calculator can handle polynomials of varying sizes without needing to allocate fixed memory.

Additionally, the Polynomials Model includes two essential methods: parsePolynomial() and getPolynomialString(). These methods serve as the backbone of the program, facilitating the conversion of user-inputted polynomials into a format stored in a HashMap, and vice versa. They form a vital link between the program's code and the user's interaction with it.

The Operations Model is the class responsible for implementing all operations such as addition, subtraction, division, multiplication, integration, and derivation. It serves as the core component for executing mathematical operations on the polynomials stored in the HashMap. These operations are crucial for performing mathematical calculations on polynomials, enabling users to manipulate and analyze polynomial expressions effectively. The Operations Model encapsulates the logic for each operation, ensuring accuracy and efficiency in computing results. Furthermore, the Operations Model provides a centralized and organized structure for managing the various mathematical functionalities required in a polynomial calculator. This approach simplifies the codebase, making it easier to maintain and expand upon in the future.

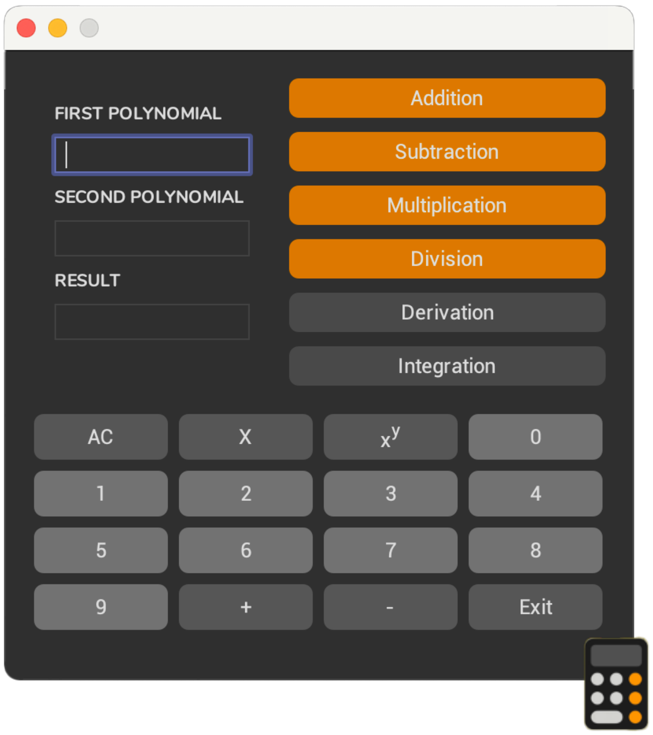
## 4. Implementation

### GUI IMPLEMENTATION

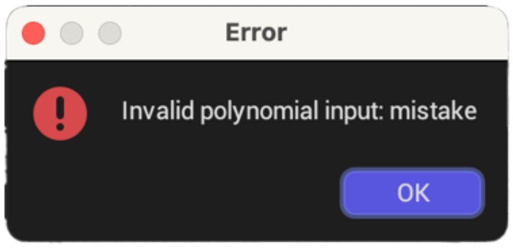
This is the visual representation of the GUI design, composed of three interconnected panels within the Calculator View Class. To elevate the visual appeal of the GUI, the FlatLaf and MigLayout add-ons are incorporated. These add-ons are included in the pom.xml file for easy integration and management.

Using the Polynomial Calculator is quite straightforward for users due to its highly user-friendly graphical interface. Upon execution of the program, the GUI initiates, and a pop-up screen emerges. The application icon also hints that this is a calculator, with the GUI design theme drawing inspiration from the native calculator in Mac OS.

Upon launch, the First Polynomial text field is automatically highlighted, allowing users to input their polynomial using either the computer keyboard or the on-screen keyboard. The polynomial should be entered in the form established by the Polynomials Controller, class that uses pattern matching in order to check the input: "ax^y+bx^z+cx^w+constant", without spaces, to facilitate parsing and conversion into a HashMap.

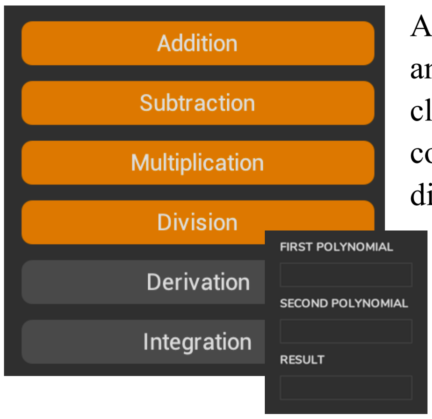


Users can select in which field to write by pressing on the respective field. There are some constraints that check that the input is valid. This Graphic Interface is implemented in the Polynomials Panel class.

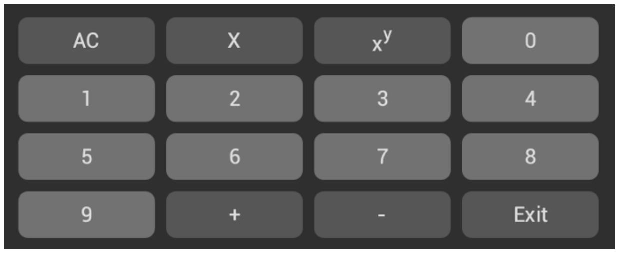


Invalid polynomial input: mistake

If there are any spaces or if the text field contains undefined characters, an error message will appear on the screen when the field loses focus. This functionality is implemented using the java.awt.event.FocusAdapter java.awt.event.FocusEvent defined in Main.java and DigitPanel.java.



After the user has inputted the correct data, they are able to select any of the Operation buttons listed in the Operations panel. Upon clicking any of these operation buttons, the corresponding computation will be executed automatically and the result will be displayed in the Result text field. This is done by using the Operations Controller. Every time an operation button is pressed, the content of the Result text field will update to reflect the new computation.



The Virtual Keyboard is integrated within the Digit Panel Class and does not require a separate Controller Class. Each button press triggers a specific action through the use of the appendText method. For buttons ranging from 0 to 9, their respective values are appended to either of the two polynomial fields in the Polynomial Panel, depending on which field is currently selected. Similarly, the buttons x, x^y, +, or - also append their respective values to the selected polynomial field. The AC button is designed to delete the text from the currently selected field, utilizing the clearText method. On the other hand, the Exit Button is intended to terminate the program entirely by executing the command System.exit(0), which exits the program with a status of 0.

## MODEL IMPLEMENTATION

The backend of the application primarily resides in the model package, which includes two key model classes: Polynomials.java and Operations.java.

### POLYNOMIALS MODEL

The Polynomials class in the org.example.model package provides functionality to work with polynomial expressions.

### CLASS DESCRIPTION:

terms: This class uses a HashMap<Integer, Integer> named `terms` to store the polynomial terms, where the key represents the exponent and the value represents the coefficient.

Constructor: Polynomials(): Initializes an empty `terms` HashMap when a new `Polynomials` object is created.

### METHODS:

addTerm(int exponent, int coefficient): Adds a new term to the polynomial with the given exponent and coefficient.

getCoefficient(int exponent): Retrieves the coefficient of the term with the specified exponent.

getPolynomialString (): Constructs and returns a readable string representation of the polynomial without spaces. The terms are ordered by exponent in descending order.

parsePolynomial(String input): Parses the input string to create the polynomial. It handles various input formats, such as "ax^y+bx^z+cx^w+constant" and updates the `terms` HashMap accordingly.

orderByDegree (): Orders the terms of the polynomial by degree in descending order.

degree(): Calculates and returns the degree of the polynomial, which is the highest exponent present in the polynomial terms.

The Polynomials class provides essential functionality to create, manipulate, and analyze polynomial expressions, making it a fundamental component of the application's backend for polynomial calculations.

OPERATIONS MODEL

The Operation class provides static methods to perform various operations on polynomial objects (Polynomials), such as addition, subtraction, multiplication, derivation, integration, and division.

METHODS:

1. public static Polynomials Addition(Polynomials poly1, Polynomials poly2)

Description: Calculates the sum of two polynomials by adding corresponding terms.

Parameters:

poly1: The first polynomial.

poly2: The second polynomial.

Returns: A new Polynomials object representing the sum of the input polynomials.

2. public static Polynomials Subtraction(Polynomials poly1, Polynomials poly2)

Description: Computes the difference between two polynomials by subtracting corresponding terms.

Parameters:

poly1: The first polynomial.

poly2: The second polynomial.

Returns: A new Polynomials object representing the result of the subtraction.

3. public static Polynomials Multiplication(Polynomials poly1, Polynomials poly2)

Description: Multiplies two polynomials by multiplying each term of the first polynomial with each term

Parameters:

poly1: The first polynomial.

poly2: The second polynomial.

Returns A new Polynomials object representing the product of the input polynomials.

4. public static Polynomials Derivation(Polynomials poly1)

Description: Calculates the derivative of a polynomial by applying the power rule.

Parameters:

poly1: The polynomial to differentiate.

Returns: A new Polynomials object representing the derivative of the input polynomial.

5. public static String Integration(Polynomials poly1)

Description: Integrates a polynomial, providing the antiderivative with a constant term "C".

Parameters:

poly1: The polynomial to integrate.

Returns: A String representing the integrated form of the polynomial with the constant term "C".

6. public static String Division(Polynomials dividend, Polynomials divisor)

Description: Divides two polynomials using polynomial long division.

Parameters:

dividend: The polynomial to be divided.

divisor: The polynomial used as the divisor.

Returns**: A String representing the quotient and remainder of the division in the format "Q: quotient, R: remainder".

7. private static Polynomials removeTermsZero(Polynomials result)

Description: Helper method to remove terms with zero coefficients from a polynomial.

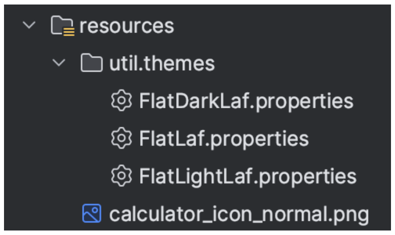
Parameters: result: The Polynomials object from which to remove zero-coefficient terms.

Returns: The modified `Polynomials` object with zero-coefficient terms removed.

The methods are designed to operate on Polynomials objects, which internally represent polynomial expressions using a HashMap<Integer, Integer> to store coefficients and exponents.

These operations allow for the manipulation and computation of polynomial expressions, which can be useful in various mathematical and scientific applications.

RESOURCES PACKAGE



```
resources
  util.themes
    FlatDarkLaf.properties
    FlatLaf.properties
    FlatLightLaf.properties
  calculator_icon_normal.png
```

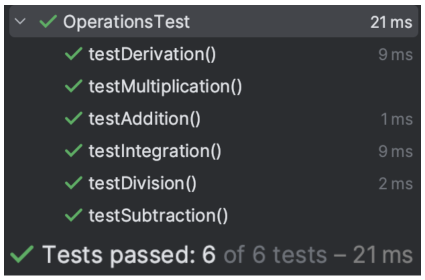
Within the resource package, there are essential items required to enhance the appearance of the GUI, including elements like the application logo to further elevate its visual appeal.

MAIN PACKAGE

The Main.java class is a Swing-based application that serves as the main entry point of a calculator program. This class sets up the main frame of the calculator application with a specific theme, adds the calculator view to the frame, and handles focus events for polynomial input text fields. It utilizes the `FlatLaf` library for modern UI design and Swing components for the user interface.

## 5. Results

By incorporating dependencies in the `pom.xml` file for JUnit, a comprehensive set of tests was created for each polynomial operation (including addition, subtraction, multiplication, division, integration, and differentiation). Each test was designed to cover various scenarios, such as negative coefficients, positive coefficients, and null coefficients. The OperationsTest class is located within the `test` package.



```
✓ OperationsTest          21 ms
    ✓ testDerivation()     9 ms
    ✓ testMultiplication()
    ✓ testAddition()       1 ms
    ✓ testIntegration()    9 ms
    ✓ testDivision()       2 ms
    ✓ testSubtraction()
✓ Tests passed: 6 of 6 tests – 21 ms
```

Upon execution, all tests were successful, confirming the accurate implementation of each operation.

## 6. Conclusions

To create an application that meets the demands of a customer, as software developers, our initial focus lies in understanding the problem through analysis and mapping it into a domain for implementation. This involves creating diagrams to facilitate a clearer understanding and implementation of the application, ensuring it aligns with user requirements. Alongside this, we must consider non-functional requirements such as performance or reliability.

Once deployed, the application can then be refined based on user feedback. This could include enhancements to the GUI and optimization of algorithms to improve performance.

Through this assignment, I have gained valuable insights into the steps involved in developing an application to meet customer needs. It introduced the use of Use Case diagrams, UML Package Diagrams, and UML Class diagrams to structure application development. I have also learned the utilization of pattern matching techniques and the process of conducting UnitTests using Junit.

## 7. Bibliography

1. For drawing Diagrams: https://app.diagrams.net/
2. Use Case Diagrams: https://www.geeksforgeeks.org/use-case-diagram/
3. More information on regex: https://regex101.com/
4. Using split command in Java: https://www.geeksforgeeks.org/split-string-java-examples/
5. Documentation on HashMap: https://www.w3schools.com/java/java_hashmap.asp