DOCUMENTATION

*ASSIGNMENT 1*

STUDENT NAME: Cornoc Roland Nicholas

GROUP: 30423

# CONTENTS

[1. Assignment Objective 3](#_Toc161977502)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc161977503)

[3. Design 8](#_Toc161977504)

[4. Implementation 12](#_Toc161977505)

[5. Results 14](#_Toc161977506)

[6. Conclusions 18](#_Toc161977507)

[7. Bibliography 19](#_Toc161977508)

# Assignment Objective

The main objective of the project is to design a polynomial calculator equal to perform basic operations such as substraction addition multiplication division integration and derivation of two polynomials and to implement a friendly graphical interface for the users.

The sub-objectives of the applications are:

* Analyze, evaluate the task, and identify the requirements.
* Design the polynomial calculator.
* Implement the polynomial calculator.
* Test the polynomial calculator.

# Problem Analysis, Modeling, Scenarios, Use Cases

**Functional and Non-Functional Requirements**

Before starting the development of any software system, it is essential to clearly define the requirements that guide its design and implementation. These requirements include both functional aspects, which determine the specific functionalities the system should provide, and non-functional aspects, which comprise qualities such as performance, reliability, and usability.

In the case of the Polynomial Calculator software application, a comprehensive understanding of both functional and non-functional requirements is crucial to ensure the system meets the needs of its users effectively.

In the following sections, we outline the functional requirements, detailing the specific operations and features the Polynomial Calculator should offer, as well as the non-functional requirements, addressing aspects such as performance, reliability, and usability that contribute to the overall quality and usability of the system.

The **Functional Requirements** of the Polynomial Calculator are:

1. Polynomial Input:

* Users should be able to input polynomial expressions into the calculator, specifying coefficients, variables, and exponents.
* The input mechanism should support standard mathematical notation for polynomials.

1. Operation Selection:

* The calculator interface should provide options for users to select the desired mathematical operation to perform on the polynomials.
* Users should be able to choose from a list of supported operations, including addition, subtraction, multiplication, division, differentiation, and integration.

1. Operations Functionality:

* The polynomial calculator should support the addition, subtraction, multiplication, division, differentiation, and integration operations, allowing users to compute polynomial expressions.
* The operations functionality should should be performed with proper handling of coefficients and exponents.

1. Result Display:

* After performing a mathematical operation, the calculator should display the result of the operation in a clear and understandable format.
* The result presentation should adhere to mathematical conventions, ensuring readability and accuracy.

1. Error Handling:

* The calculator should incorporate error handling mechanisms to detect and notify users of invalid inputs or calculations.
* Error messages should be informative and guide users in resolving issues encountered during polynomial manipulation.

The **Non-Functional Requirements** of the Polynomial Calculator are:

1. Usability:

* The Polynomial Calculator interface should be intuitive and user-friendly, allowing users to interact with the application with minimal learning curve.
* User inputs and navigation within the application should be straightforward and intuitive, enhancing overall usability.

1. Performance:

* The calculator should demonstrate optimal performance, providing prompt responses to user inputs and delivering results within acceptable time frames.
* Polynomial operations should be efficiently executed, ensuring that users experience minimal latency during calculation processes.

1. Reliability:

* The Polynomial Calculator should exhibit high reliability, performing calculations accurately and consistently under various conditions.
* The application should handle edge cases and unexpected inputs gracefully, minimizing the occurrence of errors or crashes.

**Use Cases**

The Polynomial Calculator software application offers a comprehensive array of functionalities meticulously designed to facilitate the manipulation and analysis of polynomial expressions.

With its intuitive interface and robust feature set, users can seamlessly perform various operations on polynomial functions, empowering them to tackle diverse mathematical challenges.

The use cases of the developed Polynomial Calculator software application are the following:

**Case 1. Adding Two Polynomials**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user provides the coefficients and exponents for each term of the second polynomial.
3. The user selects the option to add two polynomials.
4. The system adds the two polynomials by combining like terms.
5. The system displays the result to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a message that the polynomials do not comply with the specific format and are consequently invalid. As a result, the user should reintroduce valid polynomials.

**Case 2. Subtracting Two Polynomials**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user provides the coefficients and exponents for each term of the second polynomial.
3. The user selects the option to subtract two polynomials.
4. The system subtracts the two polynomials by performing subtraction on the terms of the two polynomials.
5. The system displays the result to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a message that the polynomials do not comply with the specific format and are consequently invalid. As a result, the user should reintroduce valid polynomials.

**Case 3. Multiplying Two Polynomials**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user provides the coefficients and exponents for each term of the second polynomial.
3. The user selects the option to multiply two polynomials.
4. The system multiplies the two polynomials by using the corresponding mathematical formula.
5. The system displays the result to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a message that the polynomials do not comply with the specific format and are consequently invalid. As a result, the user should reintroduce valid polynomials.

**Case 4. Dividing Two Polynomials**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user provides the coefficients and exponents for each term of the second polynomial.
3. The user selects the option to divide two polynomials.
4. The system divide the two polynomials by performing polynomial division algorithm.
5. The system displays the result consisting of the quotient polynomial and the remainder polynomial to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a warning message that the polynomials do not comply with the specific format and are consequently invalid. As a result, the user should reintroduce valid polynomials.
  + If the second polynomial is empty or is the zero polynomial having the degree equal with zero and the corresponding coefficient equal with zero, the system prompts an error message stating that the division cannot be performed. The user should input another polynomial as a second polynomial.

**Case 5. Derivating a Polynomial**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user selects the option to derivate the previous polynomial.
3. The system derivate the polynomial by using the derivations rules and the linear property, term by term.
4. The system displays the result to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a message that the polynomial does not comply with the specific format and is invalid. As a result, the user should input a new valid polynomial.

**Case 6. Integrating a Polynomial**

**Actors:**

* User: The person who interacts with the polynomial calculator to perform mathematical operations.

**Flow of Events:**

1. The user provides the coefficients and exponents for each term of the first polynomial.
2. The user selects the option to integrate the polynomial.
3. The system performs the integration term by term, using calculus formulas.
4. The system displays the result to the user.

**Alternative Flows:**

* + If the user inputs invalid polynomial terms, the system prompts a message that the polynomial does not comply with the specific format and is invalid. As a result, the user should reintroduce a new valid polynomial.

*O imagine care conține text, diagramă, cerc, linie

Descriere generată automat*

**Figure 1. Use Case Diagram**

# Design

In the first level of the design process, the overall system block diagram architecture is established. To this end, the inputs and the outputs of the system should be determined. The input of the system is the first polynomial, the second polynomial and the operator which selects which operation should be performed on the two polynomials. The output of the system is the result obtained by applying the operator on the two polynomials.

O imagine care conține text, Font, linie, captură de ecran

Descriere generată automat

**Figure 2. Overall System Design**

The second level of the application involves deviding the system into packages and sub-systems. In the development of this software application a 2-layered architecture was used.

GUI (Graphical User Interface) package contains classes and components responsible for user interaction, such as input forms, buttons, and result displays. It handles user input and communicates with the Business Logic package to perform operations.

Business Logic package encapsulates the core logic of the application, including polynomial operations such as addition, subtraction, multiplication, division, integration, and differentiation. It utilizes the Data Models package to represent polynomial data and performs operations based on user input received from the GUI package.

Data Models package defines the data structures and models used to represent polynomials within the application. It includes the Polynomial class and the neccesary structures to represent polynomial expressions. These models are used by both the GUI and Business Logic packages to manipulate polynomial data accurately.

O imagine care conține text, diagramă, linie, captură de ecran

Descriere generată automat

**Figure 3. Package Diagram**

In the third level of the design, the division into classes takes place. The data model to represent a proper polynomial is the class Polynomial which has as attribute a private field named monomials of type HashMap<Integer, Double>. This class encapsulates a polynomial expression using a HashMap<Integer, Double>. In this structure, the keys of the HashMap represent the exponents of the polynomial terms, and the values represent the coefficients associated with those exponents.

Using a HashMap for polynomial representation in the Polynomial class offers advantages such as efficient storage and fast lookup, as each term's exponent serves as a key for quick access to corresponding coefficients, resulting in constant-time complexity for retrieval. Additionally, HashMaps dynamically adjust their size to accommodate polynomial terms, ensuring efficient memory utilization, while their flexibility allows for the representation of polynomials with varying degrees and sparse structures, simplifying implementation and enhancing code readability.

O imagine care conține text, captură de ecran, linie, Font

Descriere generată automat

**Figure 4. Data Models Package**

The Business Logic Package contains the PolynomialUtility class which is dedicated to hosting functionalities for performing polynomial operations and providing auxiliary functions tailored for polynomial manipulation.

**O imagine care conține text, captură de ecran, linie, Font

Descriere generată automat**

**Figure 5. Business Logic Package**

The Graphical User Interface Package contains PolynomialCalculatorController class, AlertUtility class and Application class.

The PolynomialCalculatorController class acts as the central component for handling event listeners triggered by button presses related to polynomial operations.

Alongside, the AlertUtility functions to deliver informative messages to users when errors or warnings arise during application usage.

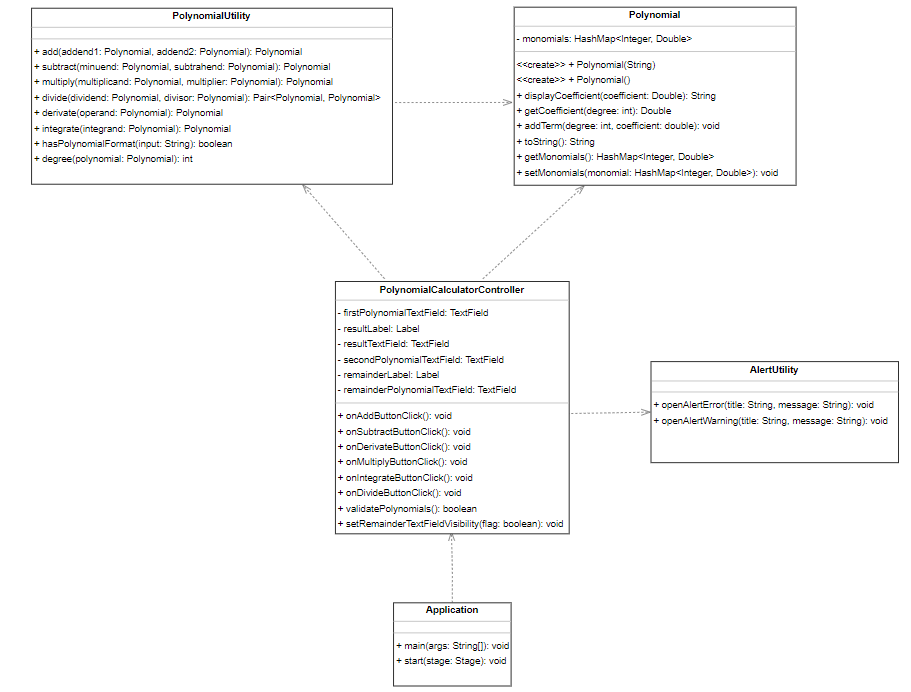
Application class serves as a controller or coordinator for initializing and managing the overall functionality of the polynomial calculator application. It might handle tasks such as setting up the user interface, initializing other components of the application, and managing the flow of control within the program.

O imagine care conține text, captură de ecran, linie, Font

Descriere generată automat

**Figure 6. Graphical User Interface Package**

In the fourth level of the design, the Class diagram is devised which contains the attributes and the methods of the class which will be implemented in the next stage. Also, the dependencies between the classes are included in the diagram.



**Figure 7. Class Diagram**

# Implementation

**Polynomial class** has a private field named **monomials** of type HashMap<Integer, Double> used to represent the monomials of the polynomial as explained in the previous section. The next methods are implemented:

* Constructor **Polynomial** initializes a Polynomial object with a given string representation.
* The method **displayCoefficient** returns the string representation of the coefficient.
* The method **getCoefficient** retrieves the coefficient corresponding to the given exponent.
* The method **addTerm** adds a new term to the polynomial with the specified exponent and coefficient.
* The method **toString** returns the string representation of the polynomial.
* The methods **getMonomials** and **setMonomials** are used for accessing and modifying the field monomials of a Polynomial object.

**O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat**

**Figure 8. Polynomial class**

The **class** **PolynomialUtility** has the following methods:

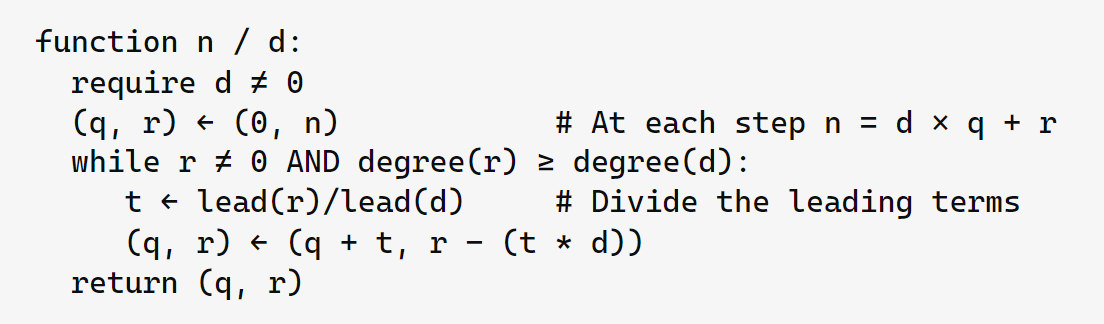
* The method **add** adds two polynomials together and returns the result.
* The method **subtract** subtracts one polynomial from another and returns the result.
* The method **multiply** multiplies two polynomials together and returns the result.
* The method **divide** divides one polynomial by another and returns the quotient and remainder as a pair.
* The method **derivate** computes the derivative of the given polynomial and returns the result.
* The method **integrate** integrates the given polynomial and returns the result.
* The method **hasPolynomialFormat** checks if the given string represents a valid polynomial format.
* The method **degree** computes and returns the degree of the given polynomial.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

**Figure 9. PolynomialUtility class**

The implementation of the **polynomial long division** algorithm is similar to performing long division with integers numbers. The leading term of the dividend is divided by the leading term of the divisor to get the first term of the quotient. Afterwards, the entire divisor is multiplied by this term and subtracted from the dividend. This process is repeated until the degree of the remainder is less than the degree of the divisor or the remainder equal to the zero polynomial.



**Figure 10. Polynomial Long Devision Pseudocode**

The methods **openAlertError** and **openAlertWarning** from the **AlertUtility class** open an error, respectively a warning alert dialog box with the specified title and message.

**O imagine care conține text, captură de ecran, Font, linie

Descriere generată automat**

**Figure 11. AlertUtility class**

In the **PolynomialCalculatorController class**, several attributes are defined to manage the user interface elements of the polynomial calculator application. The attributes include:

* **firstPolynomialTextField** component allows users to input the first polynomial for polynomial operations such as addition, subtraction, multiplication, and division.
* **secondPolynomialTextField** component, similar with firstPolynomialTextField component, enables users to input the second polynomial for polynomial operations.
* **resultLabel** component is responsible for displaying textual information about the result of polynomial operations to the user.
* **resultTextField** component displays the actual result polynomial obtained from polynomial operations.
* **remainderLabel** component is used to display textual information about the remainder of polynomial division operations.
* **remainderPolynomialTextField** component displays the remainder polynomial obtained from polynomial division operations.

Also, in the PolynomialCalculatorController class, listeners for the buttons corresponding to polynomial operations are implemented in methods **onAddButtonClick(), onSubtractButtonClick(), onMultiplyButtonClick(), onDerivateButtonClick(), onIntegrateButtonClick(),** and **onDivideButtonClick().** These methods are responsible for validating the input of the user, executing the respective polynomial operations when the corresponding buttons are clicked by the user and displaying the result.

**O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat**

**Figure 12. PolynomialCalculatorController class**

# Results

In order to ensure the correctness and robustness of the polynomial calculator application, thorough testing of its functionalities is imperative. In this section, various test cases have been devised to validate the implementation of key operations provided by the PolynomialUtility class. The JUnit testing framework has been employed to execute these tests, providing a systematic approach to validate the behavior of the polynomial calculator under different scenarios.

The tests conducted cover a range of operations including addition, subtraction, multiplication, division, differentiation, and integration of polynomials. Each test case is designed to verify the correctness of the corresponding operation, ensuring that the polynomial calculator functions as expected in various scenarios. Below are the details of each test case along with the corresponding JUnit test code snippets.

This JUnit test case verifies the functionality of polynomial addition in the Polynomial Calculator software application. Two polynomial expressions, represented by addend1 and addend2, are added together using the PolynomialUtility.add method. The expected result, representing the sum of the two polynomials, is compared against the actual result obtained from the addition operation using the assertEquals method. If the expected and actual results match, the test case passes, indicating that the polynomial addition functionality functions as expected.

O imagine care conține text, captură de ecran, Font

Descriere generată automat

**Figure 13. Add Method Test**

In the subsequent JUnit test cases, additional functionalities of the Polynomial Calculator application undergo rigorous validation to ensure robustness and accuracy. The first test examines polynomial subtraction, whereby two polynomial expressions, the minuend, and the subtrahend, are subtracted from one another using the PolynomialUtility.subtract method.

O imagine care conține text, captură de ecran, software

Descriere generată automat

**Figure 14. Subtract Method Test**

Following that, the multiplication functionality is meticulously tested, with two polynomial factors being multiplied together to yield the resulting polynomial product.

O imagine care conține text, captură de ecran, software

Descriere generată automat

**Figure 15. Multiply Method Test**

Lastly, polynomial division undergoes thorough scrutiny, wherein a dividend polynomial is divided by a divisor polynomial, and the resulting quotient and remainder are validated against expected values.

O imagine care conține text, captură de ecran, software

Descriere generată automat

**Figure 16. Divide Method Test**

The next sets of JUnit test cases evaluate the derivative and integration functionalities of the Polynomial Calculator application. Firstly, the derivative test scrutinizes the accuracy of the derivative computation for a given polynomial expression. Here, the derivative of the polynomial 3x2+2x+1 is computed using the PolynomialUtility.derivate method, and the result is verified against the expected derivative expression 6x+2.

O imagine care conține text, captură de ecran, Font, software

Descriere generată automat

**Figure 17. Derivate Method Test**

Subsequently, the integration test focuses on validating the integration operation for another polynomial expression, 6x+2. The integration of this polynomial, performed via the PolynomialUtility.integrate method, is compared against the anticipated integrated form 3x2 +2x.

O imagine care conține text, captură de ecran, Font, software

Descriere generată automat

**Figure 18. Integrate Method Test**

In addition to testing the polynomial operations, thorough validation of polynomial format inputs was conducted to ensure robustness and accuracy in the Polynomial Calculator application. Various scenarios were meticulously examined, encompassing both valid and invalid polynomial expressions. Valid polynomial expressions, including those with trailing spaces, empty strings, and properly formatted terms with complete exponents and operators, were rigorously verified using assertions to confirm their adherence to the expected format. Conversely, a comprehensive suite of tests was also devised to scrutinize a spectrum of invalid polynomial expressions, such as those with incomplete exponents, missing operators, incorrect variable notation, and non-polynomial strings. Through these extensive validation tests, the Polynomial Calculator application demonstrates its capability to handle diverse polynomial inputs effectively and accurately, thereby enhancing user experience and ensuring reliable functionality.

O imagine care conține text, captură de ecran, software

Descriere generată automat

**Figure 19. Valid Polynomials Tests**

**O imagine care conține text, captură de ecran, software

Descriere generată automat**

**Figure 20. Invalid Polynomials Tests**

# Conclusions

In conclusion, the development of the polynomial calculator application has provided valuable insights into various aspects of software design, including user interface implementation, business logic encapsulation, and integration of mathematical algorithms. Through this assignment, several key learnings have been gained.

Firstly, the importance of modular design and separation of concerns has been underscored. By organizing the application into distinct packages and classes, each responsible for specific functionalities, the codebase becomes more manageable, maintainable, and scalable. This modular approach facilitates easier debugging, testing, and future enhancements.

Secondly, the utilization of object-oriented principles, such as encapsulation, inheritance, and polymorphism, has been demonstrated in the design of classes like Polynomial, PolynomialUtility, and PolynomialCalculatorController. These principles enhance code readability, reusability, and flexibility, contributing to the overall robustness of the application.

Additionally, the implementation of algorithms for polynomial operations, such as addition, subtraction, multiplication, division, integration, and differentiation, has provided valuable exposure to mathematical computations in software development. Understanding and implementing these algorithms have deepened the understanding of mathematical concepts and their practical applications in real-world scenarios.

There are several opportunities for future developments and enhancements to the polynomial calculator application. These include:

* Improving the user interface: Enhancing the graphical user interface to make it more intuitive, user-friendly, and visually appealing.
* Adding advanced features: Introducing advanced functionalities such as polynomial plotting, symbolic computation, and support for more complex polynomial operations.
* Enhancing performance: Optimizing algorithms and data structures to improve the efficiency and speed of polynomial computations, especially for large polynomial expressions.
* Supporting additional platforms: Extending the application to run on different platforms, such as mobile devices or web browsers, to reach a wider audience.

In summary, the development of the polynomial calculator application has been a rewarding experience, offering valuable insights into software design, mathematical algorithms, and future opportunities for improvement and expansion. By leveraging the learnings from this assignment, we can continue to refine and enhance the application to meet the evolving needs of users and stakeholders.

# Bibliography

1. *Courses Materials -* [*https://dsrl.eu/courses/pt/materials/PT\_2024\_A1\_S3.pdf*](https://dsrl.eu/courses/pt/materials/PT_2024_A1_S3.pdf)
2. *Oracle Database Documentation - Regular Expression Support -* [*https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/regexp.html#GUID-1935FD80-A3CD-413F-BD2E-BBEFE64000B2*](https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/regexp.html#GUID-1935FD80-A3CD-413F-BD2E-BBEFE64000B2)
3. *GeeksforGeeks - Division Algorithm for Polynomials -* [*https://www.geeksforgeeks.org/division-algorithm-for-polynomials/*](https://www.geeksforgeeks.org/division-algorithm-for-polynomials/)
4. *Lectute Notes*