**POLYNOMIAL CALCULATOR**

**INTRODUCTION**

The application is a polynomial calculator that allows users to perform various arithmetic operations on polynomials, including addition, subtraction, multiplication, and division. It supports integration and differentiation operations as well. Users can input polynomial expressions through a graphical user interface, and the application executes the specified operations, displaying the results. The application utilizes object-oriented programming principles, such as encapsulation and modular design, to achieve maintainability and extensibility. Additionally, it employs the Singleton design pattern to manage a global variable and ensures consistency throughout the application. Overall, the application provides a user-friendly interface for performing polynomial calculations efficiently.

The application consists of several classes and interfaces to facilitate polynomial operations and user interactions:

**Polynomial Class**: Represents a polynomial expression and provides methods for manipulating polynomials, such as adding, subtracting, multiplying, dividing, integrating, and differentiating.

**Monomial Class**: Represents a single term within a polynomial, containing a coefficient and an exponent.

**Arithmetic Interface**: Defines methods for performing arithmetic operations on polynomials, including addition, subtraction, multiplication, division, integration, and differentiation.

**Transformation Interface**: Defines methods for transforming polynomial expressions, such as parsing strings into polynomial objects and converting polynomial objects to string representations.

**OperationType Enum**: Enumerates different types of polynomial operations, such as ADD, SUBTRACT, MULTIPLY, DIVIDE, INTEGRATE, and DIFFERENTIATE.

**Variable Class:** Implements the Singleton design pattern to manage a global variable used in polynomial calculations.

**SuperScript Class**: Provides a utility method to convert numbers into superscript Unicode characters for displaying exponents in polynomial expressions.

**OperationsHandling Class**: Orchestrates polynomial operations based on user input, utilizing arithmetic and transformation services, and handles error messages and display logic.

**Interface Class**: Implements a graphical user interface using Swing components to allow users to input polynomial expressions and view results.

**The Polynomial class** represents a polynomial expression composed of multiple monomials. Here's an explanation of its functionality:

**Terms Map (terms):** The polynomial is represented using a navigable map (NavigableMap) where the keys represent the degrees of the monomials and the values represent the corresponding monomials. The map is sorted in reverse order based on the degrees of the monomials.

**getHigherDegree()** method: This method returns the highest degree of the polynomial by retrieving the first entry from the terms map. If the polynomial is empty, it returns 0.

**getTerms()** method: This method returns the navigable map containing the terms of the polynomial.

**getVariable()** method: This method returns the global variable associated with the monomials in the polynomial.

**The Monimial class** represents a single term in a polynomial expression. Each monomial consists of three main components:

**Coefficient (coeff):** This represents the numerical coefficient of the monomial, indicating the scale or magnitude of the term.

**Variable (var):** This represents the variable associated with the monomial. It could be any symbol or letter representing a quantity that can vary.

**Exponent (expo):** This represents the power to which the variable is raised in the monomial, indicating the degree or order of the term within the polynomial.

**The Arithmetic interface** defines a set of operations that can be performed on polynomial expressions. Here's an explanation of its methods:

**add(Polynomial p1, Polynomial p2):** This method computes the sum of two polynomial expressions p1 and p2 and returns the result as a new polynomial.

**subtract(Polynomial p1, Polynomial p2):** This method computes the difference between two polynomial expressions p1 and p2 and returns the result as a new polynomial.

**multiply(Polynomial p1, Polynomial p2):** This method computes the product of two polynomial expressions p1 and p2 and returns the result as a new polynomial.

**divide(Polynomial p1, Polynomial p2):** This method computes the division of polynomial expression p1 by polynomial expression p2. It returns a hashmap containing three polynomials: the result of the division (result), the remainder (reminder), and the divisor (divisor). The key-value pairs are stored in a hashmap where the keys are strings ("result", "reminder", "divisor") and the values are the corresponding polynomials.

**integrate(Polynomial p):** This method computes the indefinite integral of a polynomial expression p and returns the result as a new polynomial.

**differentiate(Polynomial p):** This method computes the derivative of a polynomial expression p and returns the result as a new polynomial.

**The Transformation interface** provides a set of methods for transforming polynomial expressions between different representations. Here's an explanation of its methods:

**parse(String inputString):** This method takes a string representation of a polynomial expression as input and parses it into a Polynomial object. It converts the string representation of the polynomial into its corresponding polynomial object representation.

**convertToString(Polynomial polynomial):** This method takes a Polynomial object as input and converts it into a string representation. It generates a string that represents the polynomial expression in a human-readable format.

**The OperationType enum** defines different types of polynomial operations, each associated with a symbolic representation. Here's an explanation of its components:

**Enum Constants:**

ADD, SUBTRACT, MULTIPLY, DIVIDE, INTEGRATE, DIFFERENTIATE: These enum constants represent different types of polynomial operations such as addition, subtraction, multiplication, division, integration, and differentiation. Each enum constant is associated with a specific symbolic representation.

**fromString(String str):** This static method converts a string representation of an operation type into its corresponding enum constant. It iterates through all enum constants and returns the one whose symbolic representation matches the input string. If no matching enum constant is found, it throws an IllegalArgumentException.

**The Interface class** represents a Swing-based user interface for a polynomial calculator application. Here's an explanation of its functionality:

**User Interface Components:**

**JTextField:** Input fields for entering polynomial expressions and displaying results.

**JButton:** Buttons for performing arithmetic operations (addition, subtraction, multiplication, division, integration, differentiation) and clearing the input fields.

**JPanel:** Panels for organizing and grouping components.

**JLabel:** Labels for indicating the purpose of input fields.

**Constructor:**

Initializes the frame and sets its properties such as title, size, and layout.

Creates input fields, buttons, and panels.

Adds components to the main panel and sets the layout.

**Action Listeners:**

**OperatorButtonListener**: Listens for button clicks on arithmetic operation buttons. Retrieves polynomial expressions from input fields, determines the operation type based on the button clicked, executes the corresponding operation using OperationsHandling, and displays the result in the result field.

**ClearButtonListener**: Listens for button clicks on the clear button. Clears the input fields and result field.

**Utility Methods:**

**getOperationType(String actionCommand):** Converts the action command of a button into an OperationType enum constant.

**setComponentMargins(Component component):** Sets margins for buttons to improve visual appearance.

**The SuperScript class** provides a utility method to convert a number into a string of Unicode characters representing superscript digits. Here's an explanation of its functionality:

**convert(int number):** This method takes an integer number as input and converts it into a string of superscript Unicode characters. It first initializes an array digits containing Unicode representations of superscript digits from 0 to 9. Then, it iterates over each digit of the input number, retrieves its Unicode representation from the digits array, and appends it to a StringBuilder. Finally, it returns the resulting string of superscript characters.

**The OperationsHandling class** serves as a service layer responsible for executing polynomial operations based on user input and handling the display of results or errors. Here's an explanation of its functionality:

**execute():** This method performs the specified operation (operationType) on the given polynomial expressions (firstPolyString and secondPolyString). It first initializes instances of PolynomialTransformation and PolynomialArithmetic for transformation and arithmetic operations, respectively. Then, it parses the input polynomial expressions into Polynomial objects using the transformation service. Depending on the operation type, it either performs integration, differentiation, or arithmetic operations (addition, subtraction, multiplication, division). The result of the operation is converted back into a string representation using the transformation service.

**handleDivision():** This method handles division operations separately due to the complexity of handling divisors. It computes the quotient, remainder, and divisor using the arithmetic service and then converts them into string representations using the transformation service.

**displayError():** This method displays an error message to the user. It uses a dialog box from the Swing library (JOptionPane) to show the error message. This method is used to inform the user about invalid input or errors encountered during operation execution.