TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

Polynomial Calculator

Assignment 1

- Documentation -

Student: Alexandra Ilovan
Group: 30422
Laboratory Professor: Andreea-Valeria Vesa

TABLE OF CONTENTS

1. Assignment Objectives

The main objective of the given assignment is to design and implement a polynomial calculator with a dedicated user interface through which the user can insert polynomials, select the mathematical operation to be performed (the user can choose between addition, subtraction, multiplication, division, derivation or integration) and view the result.

Among the secondary objectives of the assignment are the following:

- analyzing the problem and identifying the requirements
- implementing efficient algorithms for each operation
- designing an intuitive graphical user interface that is easy to use
- implementing a functional polynomial calculator
- generating correct results for the given operations
- displaying the results on the graphical interface

2. Problem Analysis. Modeling. Scenarios and Use Cases

2.1. Problem Analysis

Stating the problem: "Performing polynomial operations on paper is often difficult and time consuming."
The solution to this problem is to implement a polynomial calculator that allows the user to enter two polynomials of choice, select the desired operation to be performed on the given polynomials, and provide correct results in a matter of seconds.

2.2. Modeling

In order to understand the process of modeling entirely, one has to first be familiarized with the theoretical background of polynomials.
A polynomial P in an indeterminate X is an expression formally defined as:

$$P(X) = a_n \cdot X^n + a_{n-1} \cdot X^{n-1} + \ldots + a_1 \cdot X + a_0$$

where:

- $a_1, a_2, \ldots, a_n$ represent the polynomial's coefficients
- n represents the degree of the polynomial
- X represents the term of the polynomial

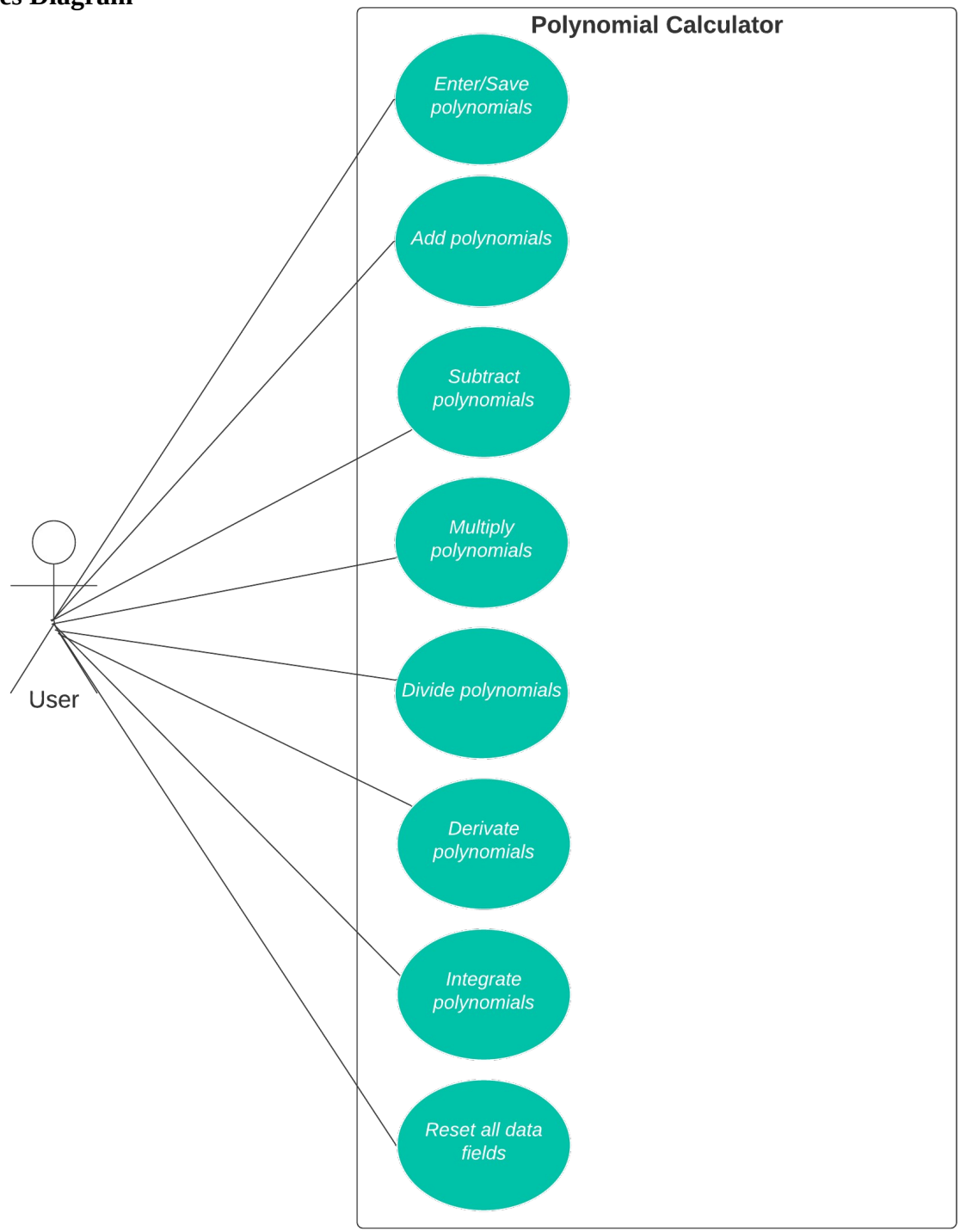
The implementation of a polynomial in this assignment, however, also required the understanding of the concept of monomials. A monomial is a special type of polynomial that has only one term. In the expression of the polynomial P(X) from above, the structure $a_n \cdot X^n$ represents a monomial, for instance. So in order to replicate the model of a polynomial, a list of monomials (structures containing a coefficient and a degree) had to be defined.

2.3. Scenarios and Use Cases

The polynomial calculator can receive either one or two polynomials as input, depending on the operation that has to be performed. For instance, if the user chooses to perform the operation of addition, subtraction, multiplication or division, entering two polynomials will be required. Instead, if the chosen operation is derivation or integration, those can only be performed on one polynomial. The monomials can be entered in any order, since the input will be sorted automatically by degree. In the case when the input has been entered incorrectly, the application will display a message stating that the input is not valid, and that no operations can be performed. After entering any of the polynomials, the user has the option to either save the data or clear that field. Upon saving the data, the operation to be

performed can be selected by pressing the corresponding button, and depending on the operation, the result will be displayed under the fields where the polynomials have been entered. Besides performing calculus on polynomials, the user is also provided the option to reset all the data, or exit the application.

Use Cases Diagram



3. Design

3.1. Design Pattern

The design of the application is based on the **MVC** (Model-View-Controller) architecture, containing three main packages inspired by it: **controller**, **model** and **view**.

The **view** package consists of a class **CalculatorView**, which is used for presenting the data of the model to the user. It deals with the positioning of components such as JTextFields, Jbuttons and adding functionality to those (which is done by creating corresponding Action Listeners). A file with the same name as the class mentioned before but with the .form extension can also be found inside the view package, and it contains a translation in code of all the component arrangement done using the GUI designer provided by Swing.

The **controller** package contains the class **CalculatorController**, which acts as a link between the view and the model. It listens to all the actions triggered in the view (such as keyboard input, button pressing, the activation of mouse movement etc.) and performs an appropriate response back to the events by calling methods found in the model.

The **model** package, which encapsulates the core data and functionality, is made out of 5 classes: the **Polynomial** and **Monomial** classes are used for defining and storing the polynomials in a usable way; the **Operation** class implements methods for all the operations that can be performed on polynomials; and the **StringHandler** and **InvalidUserInputException** are classes that contain methods used for extracting polynomials from a given String and for checking that the input is valid, which is only made possible with the help of regular expressions and pattern matching.

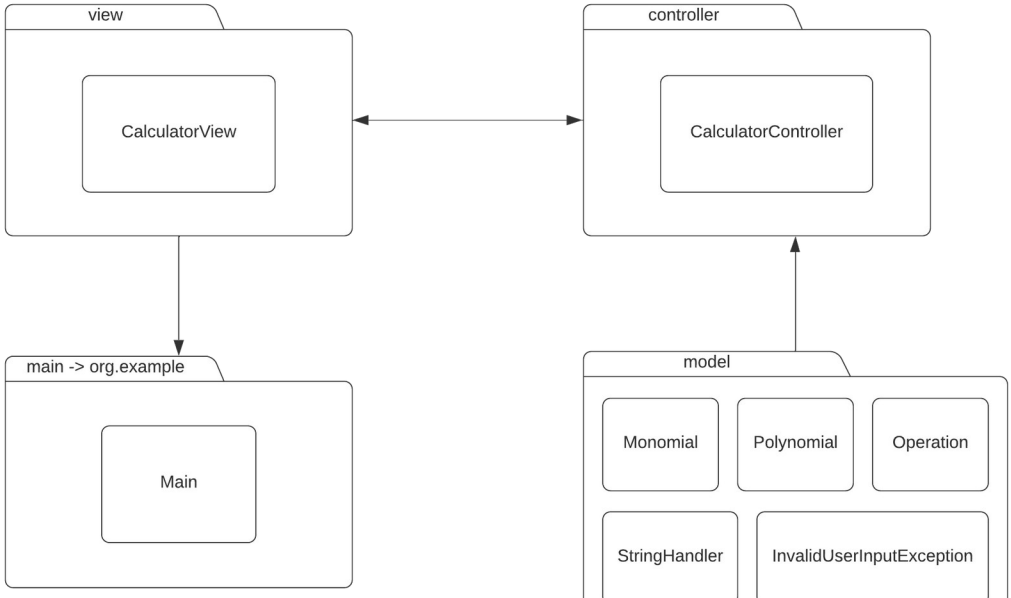
The **org.example** package contains the **Main** class, where an object of type CalculatorView is being instantiated in order to run and display the application.

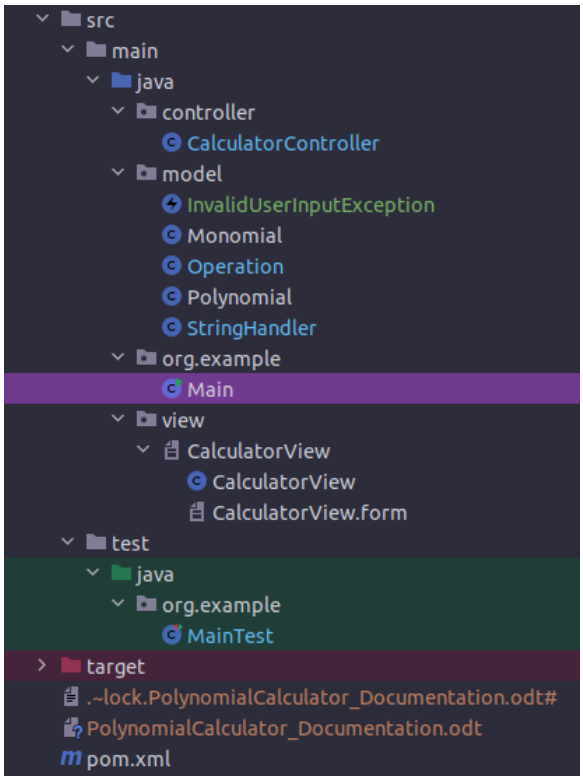
Another aspect worth mentioning is the inclusion of the Comparable interface, which is implemented by the Monomial class in order to perform the sorting operation of the list of monomials that represent the polynomial structure.

The project also contains a class **MainTest**, which performs JUnit testing and checks the correctness and accuracy of all the operations that can be performed, providing at least two tests for each one. In case one operation is incorrect, an error message saying that the test has failed will be displayed.

3.2. Package Diagram

A diagram presenting the relationships between the packages explained above is attached below:



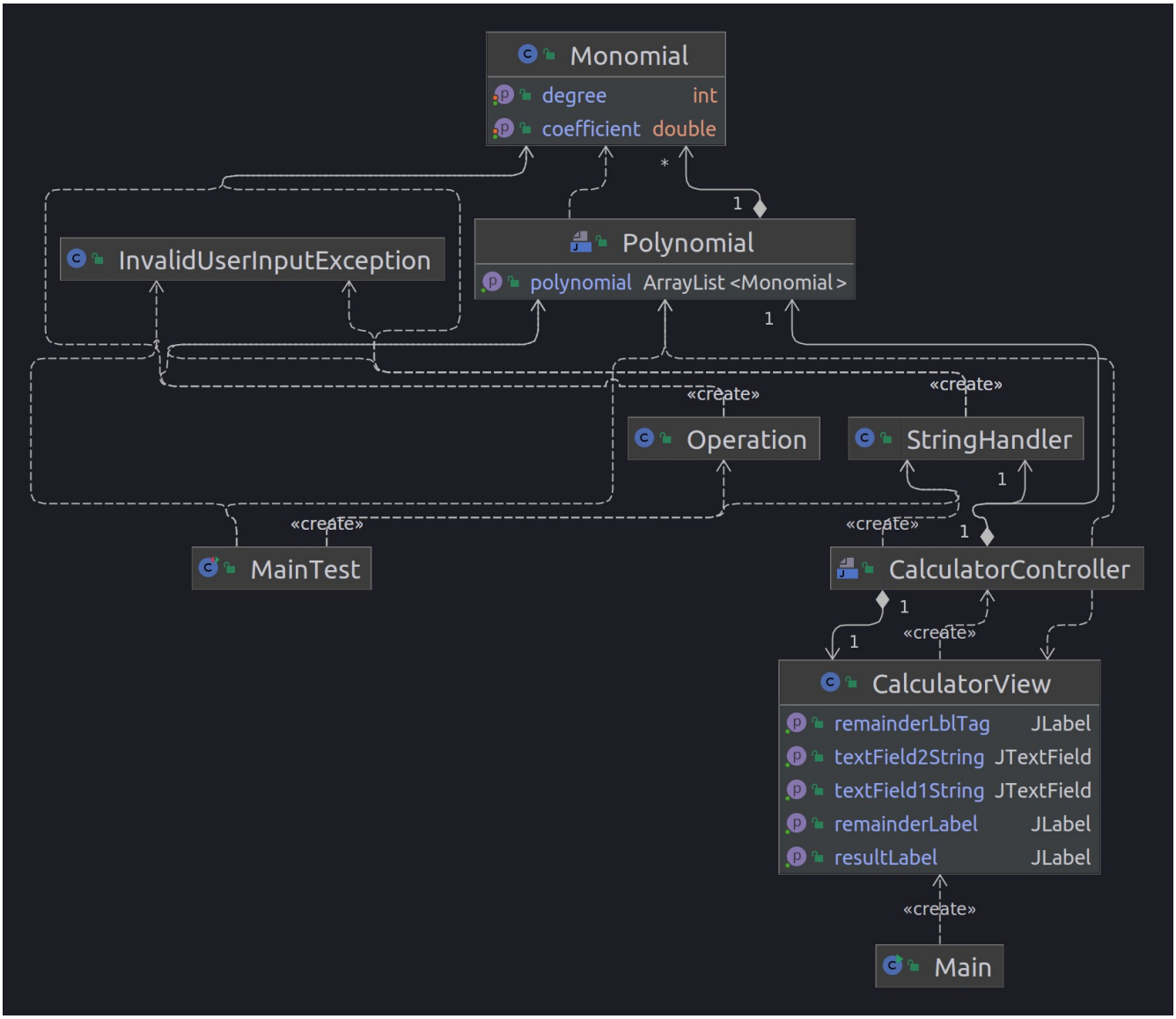


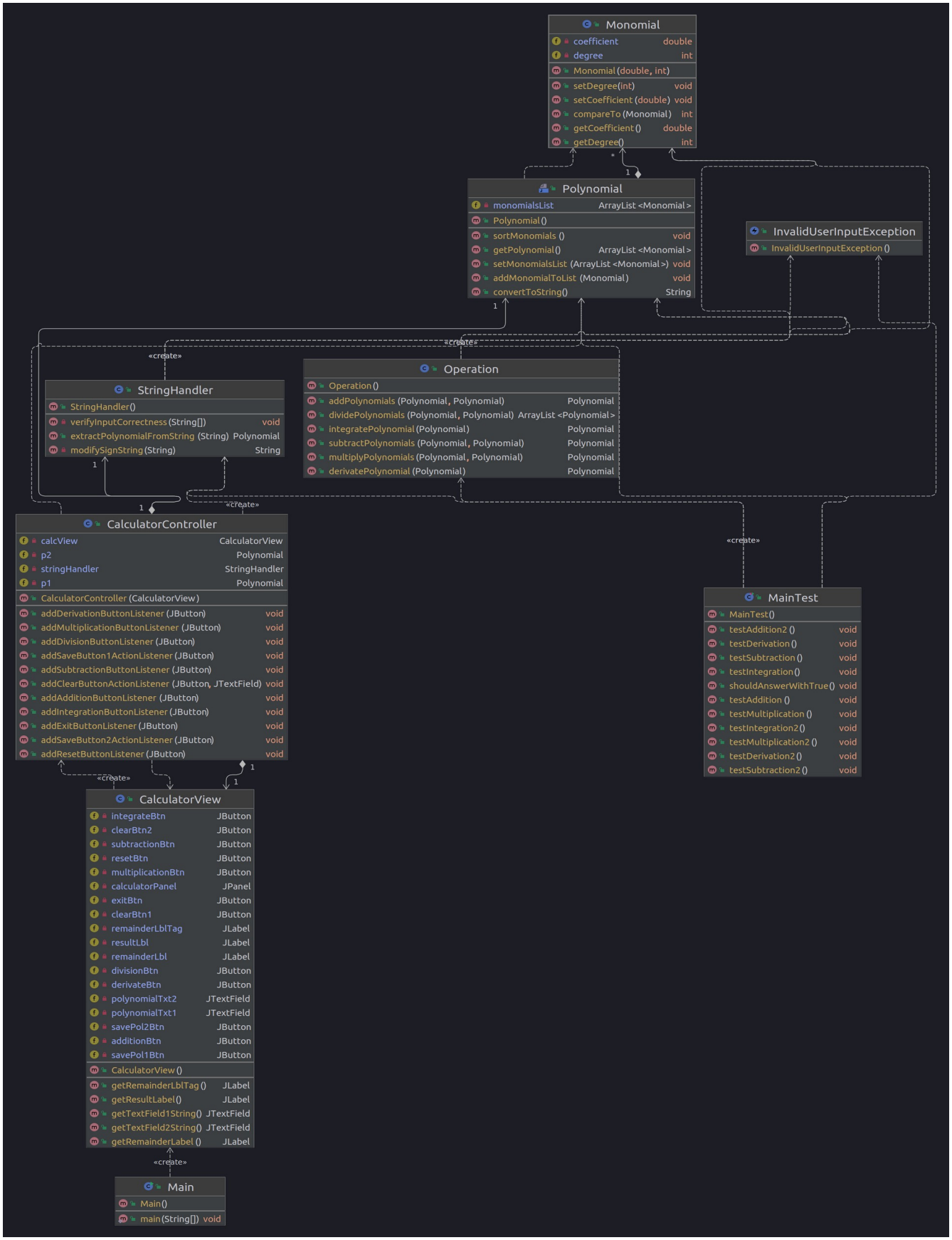
3.3. UML Diagram

The UML Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

- classes
- attributes of classes
- operation (or methods)
- relationships between objects/classes

The UML class diagram generated by IntelliJ is represented on the next page:




Monomial
coefficient double
degree int
Monomial(double, int)
setDegree(int) void
setCoefficient(double) void
compareTo(Monomial) int
getCoefficient() double
getDegree() int
Polynomial
monomialsList ArrayList<Monomial>
Polynomial()
sortMonomials() void
getPolynomial() ArrayList<Monomial>
setMonomialsList(ArrayList<Monomial>) void
addMonomialToList(Monomial) void
convertToString() String
InvalidUserInputException
InvalidUserInputException()
«create»
StringHandler
StringHandler()
verifyInputCorrectness(String[]) void
extractPolynomialFromString(String) Polynomial
modifySignString(String) String
Operation
Operation()
addPolynomials(Polynomial, Polynomial) Polynomial
dividePolynomials(Polynomial, Polynomial) ArrayList<Polynomial>
integratePolynomial(Polynomial) Polynomial
subtractPolynomials(Polynomial, Polynomial) Polynomial
multiplyPolynomials(Polynomial, Polynomial) Polynomial
derivatePolynomial(Polynomial) Polynomial
CalculatorController
calcView CalculatorView
p2 Polynomial
stringHandler StringHandler
p1 Polynomial
CalculatorController(CalculatorView)
addDerivationButtonListener(JButton) void
addMultiplicationButtonListener(JButton) void
addDivisionButtonListener(JButton) void
addSaveButton1ActionListener(JButton) void
addSubtractionButtonListener(JButton) void
addClearButtonActionListener(JButton, JTextField) void
addAdditionButtonListener(JButton) void
addIntegrationButtonListener(JButton) void
addExitButtonListener(JButton) void
addSaveButton2ActionListener(JButton) void
addResetButtonListener(JButton) void
MainTest
MainTest()
testAddition2() void
testDerivation() void
testSubtraction() void
testIntegration() void
shouldAnswerWithTrue() void
testAddition() void
testMultiplication() void
testIntegration2() void
testMultiplication2() void
testDerivation2() void
testSubtraction2() void
CalculatorView
integrateBtn JButton
clearBtn2 JButton
subtractionBtn JButton
resetBtn JButton
multiplicationBtn JButton
calculatorPanel JPanel
exitBtn JButton
clearBtn1 JButton
remainderLblTag JLabel
resultLbl JLabel
remainderLbl JLabel
divisionBtn JButton
derivateBtn JButton
polynomialTxt2 JTextField
polynomialTxt1 JTextField
savePol2Btn JButton
additionBtn JButton
savePol1Btn JButton
CalculatorView()
getRemainderLblTag() JLabel
getResultLabel() JLabel
getTextField1String() JTextField
getTextField2String() JTextField
getRemainderLabel() JLabel
Main
Main()
main(String[]) void

4. Implementation

The implementation breakdown will be categorized by packages, as follows:

- **view** package
  - **CalculatorView** class: This class contains as main attributes several JButtons of various usages (there are 6 buttons for performing the polynomial operations, 2 save buttons – used for storing the polynomial input data extracted from the text fields, 2 clear buttons – used to clear the text fields, one reset button and one exit button), two JTextFields used for the extraction of input data entered by the user, two JLabels – one for displaying the results and the second for the remainder (in the case of the division operation).
    The CalculatorView class extends JFrame, and its constructor contains the set-up needed to launch the window on the screen along with an instance of the CalculatorController class, so that the dependence between the components and their functionality is being established through their action listeners.

- **controller** package
  - **CalculatorController** class: This class' attributes include an instance of the class CalculatorView, in order to be able to have access to and manipulate the components from the view, and an instance of the class StringHandler, so that polynomials can be extracted from the user input correctly. Moreover, the class implements methods for adding Action Listeners for each component of the view that is relevant or is being triggered by the user in any way (operation buttons, save/ reset/ clear/ exit buttons).

- **model** package
  - **Monomial** class: The attributes of the Monomial class are a type double coefficient (it is required that the coefficient has a floating point type so that the integration result will be accurate), and the degree of the monomial, of type int. Besides a constructor, getters and setters, this class also implements the Comparable interface and overrides the method compareTo() in order to facilitate the sorting of the monomials in descending order of their degrees.
  - **Polynomial** class: Its attribute consists solely of an ArrayList of monomials. The Polyomial class implements 4 methods: one that adds a new monomial to the existing list - *addMonomialToList(Monomial monomial),* one that returns the current list of monomials – *getPolynomial()*, one that sorts the monomials in the list in descending order of their degrees – *sortMonomials()*, and one that realizes the conversion from Polynomial to String – *convertToString()*. The latter is explained in the following image:

```java
// method that converts a polynomial to its string form
public String convertToString()
{
    final String[] polynomialString = {""};

    if (this.monomialsList.isEmpty())
        polynomialString[0] = "-";  // if the list of monomials is empty, the resulting string will only contain "-"

    // if the list is not empty, we traverse it
    monomialsList.forEach(monomial -> {
        if (monomial == monomialsList.get(0))
        {
            // converting the coefficient
            if (monomial.getCoefficient() != 0) // if the coefficient is not null
            {
                if (monomial.getCoefficient() == -1 && monomial.getDegree() > 0)  // if the coefficient is -1 and the degree is positive
                    polynomialString[0] += "-"; // we add the minus sign preceding the monomial
                else if (monomial.getCoefficient() != 1)   // if the coefficient is anything but 1, we add the number of the coefficient
                    polynomialString[0] += monomial.getCoefficient();   // to the string
                else if (monomial.getCoefficient() == 1 && monomial.getDegree() == 0)   // if the coefficient is 1 and we have x^0
                    polynomialString[0] += "1"; // add only the value 1
            }
        }
        else
        {
            if (monomial.getCoefficient() != 0)
            {
                // treating the cases where we have a positive coefficient
                if (monomial.getCoefficient() == 1 && monomial.getDegree() != 0)
                    polynomialString[0] += "+";
                else if (monomial.getCoefficient() > 0)
                    polynomialString[0] += "+" + monomial.getCoefficient();
                else if (monomial.getCoefficient() == -1 && monomial.getDegree() > 0)   // and the case where it's negative
                    polynomialString[0] += "-";
                else
                    polynomialString[0] += monomial.getCoefficient();
            }
        }

        // converting the degree
        if (monomial.getDegree() != 0 && monomial.getCoefficient() != 0)
        {
            if (monomial.getDegree() == 1)  // if the degree is 1, only "x" will be displayed
                polynomialString[0] += "x";
            else
                polynomialString[0] += "x^" + monomial.getDegree();
        }
    });

    return polynomialString[0];
}
```

➔ **Operation** class: This class contains the methods corresponding to every operation that can be performed on polynomials. The methods responsible for performing the addition, subtraction and multiplication operations receive as input parameters two polynomials and return a polynomial as well, while the division method receives as parameters two polynomials but returns an ArrayList of Polynomials (where the first element of the list represents the quotient of the division, and the second element is the remainder). The derivation and integration are the only operations whose methods only take one polynomial as input parameter and return a polynomial for the result.

The addition and subtraction operations (the methods addPolynomials() and subtractPolynomials() respectively) have a similar implementation (only the signs differ). In order to compute the result, a merge-like manner is used. The two monomial lists corresponding to the two polynomials are iterated through with different iterators, going from the monomial with the highest degree towards the one with the smallest degree. The degrees of the corresponding monomials are compared and, in the case in which the degree of one of the monomials is larger than the other's, that monomial is added to the result list and its iterator is incremented. If the degrees are equal, however, depending on the operation, a monomial containing either the sum/difference of the coefficients and the given degree is added to the result list and both of the iterators are incremented.
The multiplication operation is computed through the successive traversal of each polynomial in a monomial-by-monomial manner, where each monomial is multiplied with all the monomials of the other polynomial in order to generate a new polynomial – which would represent the multiplication result.
The division operation is done the following way:

- say we have two polynomials P and Q
- suppose we want to compute P/Q
- first, the monomial with the largest degree of P is divided by the monomial with the maximum degree of Q
- the resulting polynomial (the quotient) will then be multiplied by the polynomial Q
- we subtract from polynomial P the result obtained previously and the result will represent the remainder
- then repeat the previous steps until the maximum degree of the remainder is smaller than the maximum degree of Q

The derivation operation is done by traversing the list of monomials, by multiplying the coefficient of the current monomial with its degree for the resultig monomial's coefficient and by decrementing the degree of the current monomial by one. The integration operation is the inverse of the derivation: the degree is incremented by one, and the coefficient is divided by that value.

- **InvalidUserInputException** class: It extends the Exception class, and is used only if the user enter invalid data for the input. The methods will throw this exception if they encounter incorrect inputs.

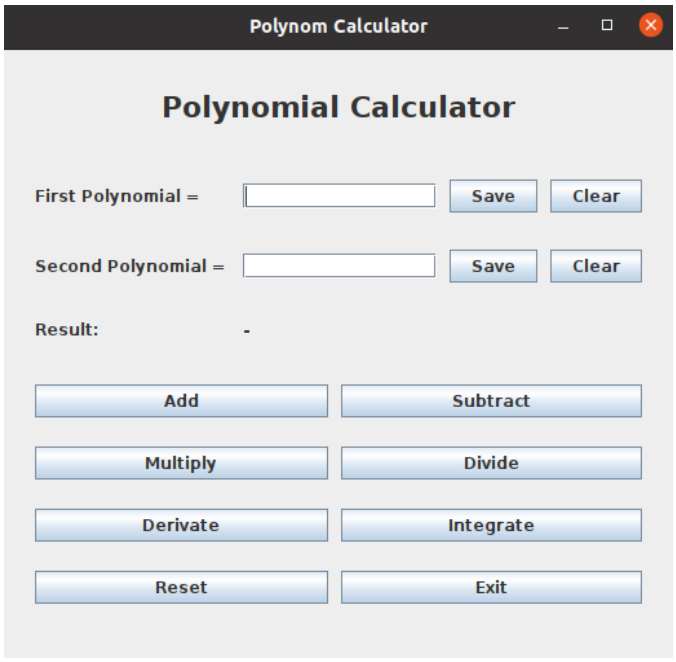
- **StringHandler** class: This class contains one public method that deals with extracting a polynomial structure from a String and throws an InvalidUserInputException is the given string is not valid → *extractPolynomialFromString(String polynomialString)*. However, the class also contains two private methods that are used in the extraction method described previously: *modifySignString(String initialStr)* – a method that replaces the character "-" with the sequence "+-" every time it encounters it in a string, and *verifyInputCorrectness(String[] input)* – method that uses regular expressions and pattern matching to identify whether a given string is written in a valid manner or not, and in the case that it doesn't, it throws the exception defined above. The conversion from String to Polynomial is done by following the next steps:
  - the input string is being processed and each occurrence of the "–" sign is replaced by "+-"
  - the string is splitted into an array of strings, and the separation is done after every + sign
  - each splitted string will be processed individually and will generate a monomial for the list

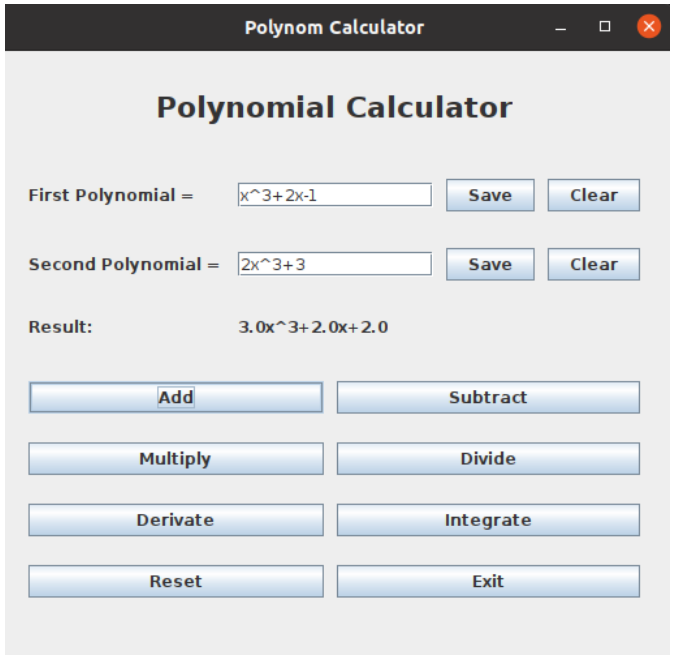
5. Results

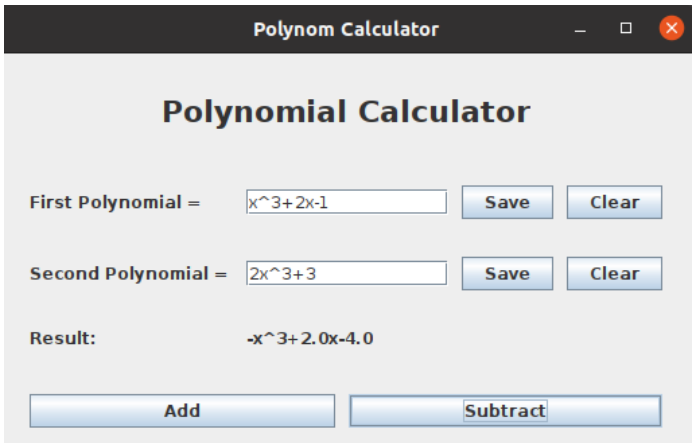
The results can be tested either by using JUnit testig, or by using the graphical interface.

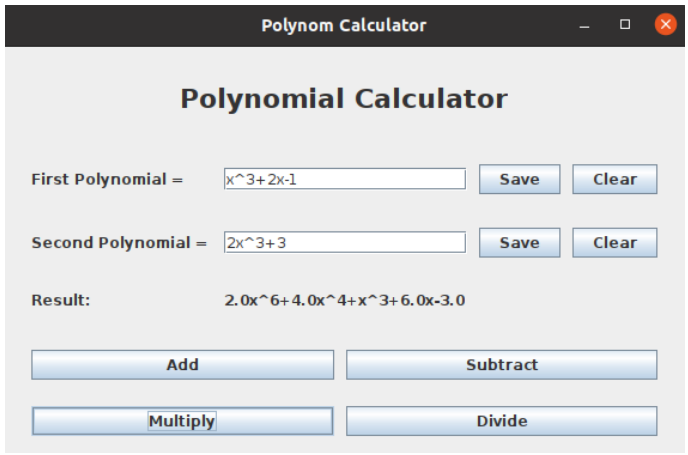
For the **JUnit testing**, 10 methods that test each operation on different sets of polynomials were defined inside the **MainTest** class. In order to verify another set of polynomials besides the ones that I have already introduced, both the strings representing the polynomials inside the methods and the string that the result is being compared to have to be modified.

In the case of testing by the usage of the GUI, the user can enter at most two polynomials in the specified text fields, save their values, and press the button corresponding to the operation he/she wants to perform.

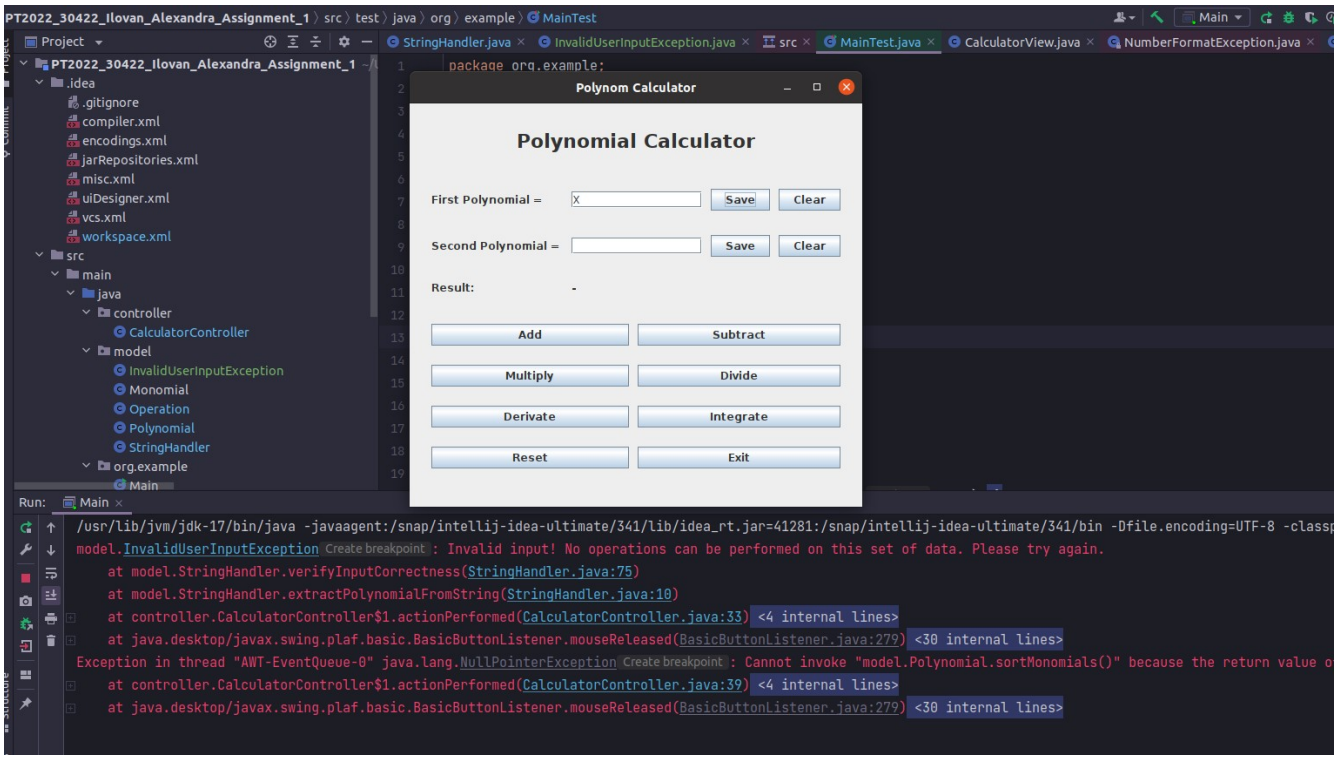








When entering invalid input:



6. Conclusions

By working on this assignment, I have gained a better understanding of managing the MVC architectural pattern, and had the opportunity to dive deeper into learning about regular expressions. Pattern matching is still a bit of a challenge to me, but this project helped me take a step further.
This was also the first time I used JUnit testing, and to my pleasant surprise, I found it especially useful when I had to implement new features, and saved the time of testing it manually in the GUI by having all the tests already written and ready to be checked.

7. Bibliography

- The provided support presentation of the current assignment
- The PT courses
- https://www.w3schools.in/mvc-architecture/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
- https://www.vogella.com/tutorials/JUnit/article.html
- https://www.tutorialspoint.com/java/java_regular_expressions.htm
- https://stackoverflow.com/
- used LucidChart for the Use Cases and Package diagrams