# FUNDAMENTAL PROGRAMMING TECHNIQUES

Assignment 1

# Polynomial Calculator

Student: Halmai Erik

Group: 30422

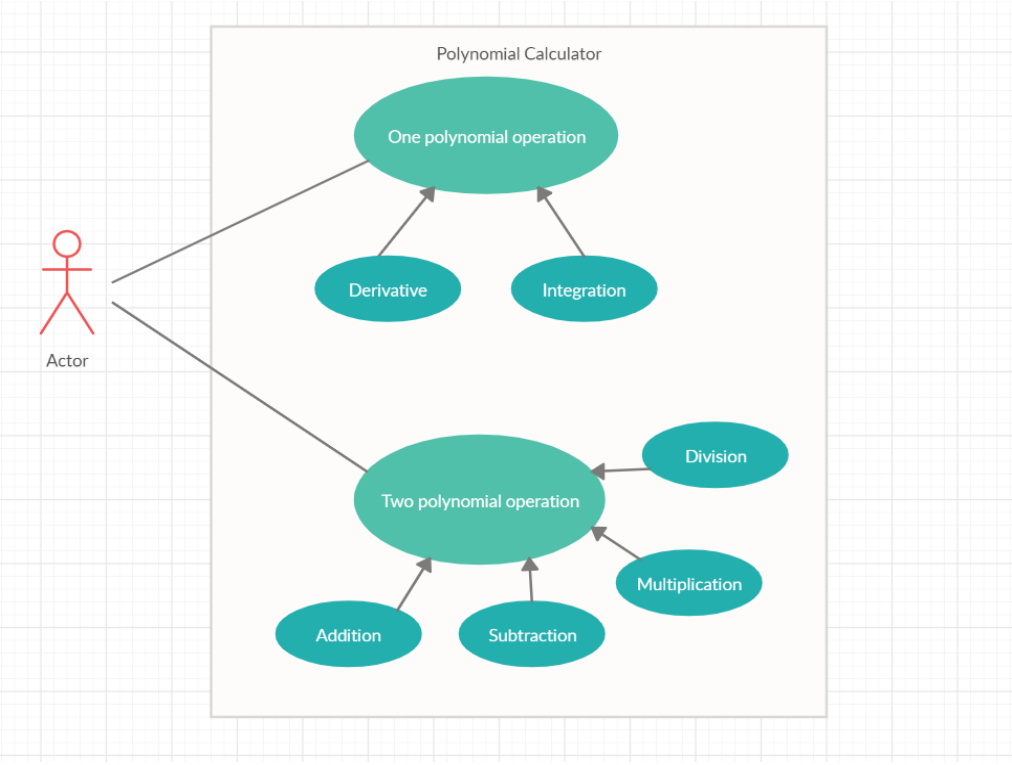
# 1. Objective

The main objective of the assignment is the creation of a polynomial calculator with dedicated graphical interface through which the user can enter polynomials and choose the desired operation to be performed (derivative, integration, addition, subtraction, multiplication or division).

The system's main objective of the assignment can be divided into several secondary objectives, each achieved separately. These secondary objectives represent the steps that are taken in order to achieve the main objective of the assignment. These steps are the following:

1) **Creation of an object that can store the polynomials (Chapter 4)**: The user entered polynomials need to be split up into the smallest part of the polynomial: the monomial, which then can be stored in a list.
2) **Validation (Chapter 4)**: The polynomials entered by the user need to be checked for any character that may cause the program to crash. This is done using regular expressions and pattern matching.
3) **Performing the desired operation (Chapter 4)**: The polynomial operations can be divided into two groups: the ones working with one polynomial, as derivative and integration, and the ones working with two polynomials, as addition, subtraction, multiplication and division. The result of these operations have to be sorted and simplified.
4) **Handling unwanted or unplanned situation (Chapter 4)**: Handling the exceptions is a big part of the project. The main goal is the creation of an application that doesn't crash when it runs into exceptions or errors, but rather notifies the user to correct their mistakes.
5) **Construction of the GUI (Chapter 4)**: The graphical interface is designed according to the Model View Controller architectural pattern.

# 2. Problem analysis, modeling, scenarios, use cases

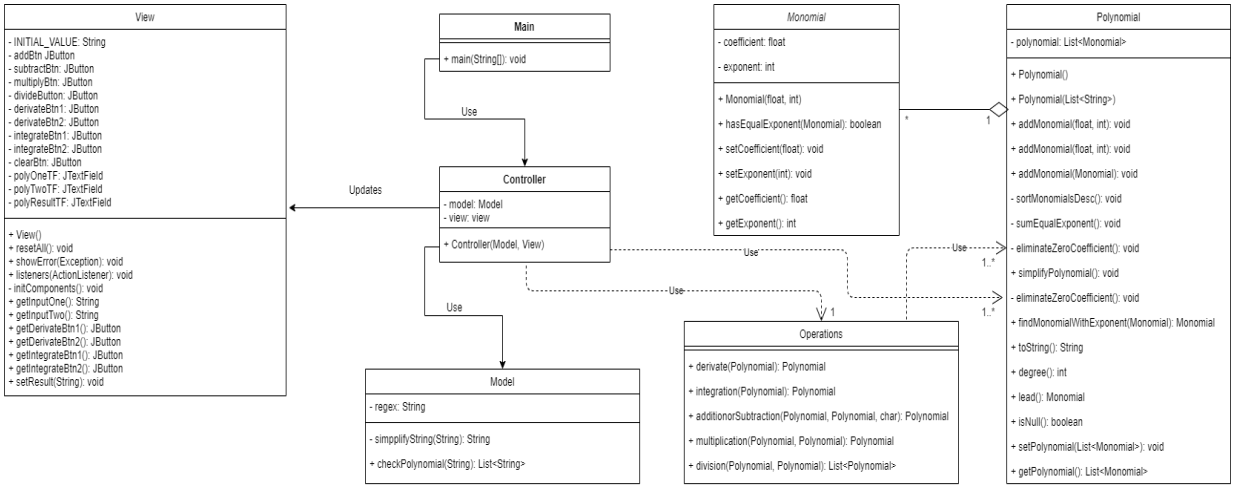


1) **Introducing the input:** The polynomials used by this application need to be in the following format: **coefficient*x^exponent**, the coefficient being a float and the exponent a positive integer number. The coefficient and the exponent value can be emitted when equal to +-1 and only the coefficient's value can be written when the exponent is 0. The user can enter the polynomial with or without spaces and '*' character. No other characters are allowed in the input string, else an error message appears. When launching the application, the fields are set to "0", but if the user decides to perform an operation with an empty input field, another error message appears. This validation process, and the split up of the polynomial into monomials is done using regex and pattern matching.

2) **Selecting the operation:** The user can select from 6 possible operations: derivative, integration, addition, subtraction, multiplication or division. Two of these six operations (derivative and integration) work on a single operand. Two buttons are located next to the two input fields, which perform these operations on the field that they are next to. The other four operations need two fields. These are marked with buttons having '+' for addition, '-' for subtraction, '*' for multiplication and '/' for division.

3) **Getting the result:** When launching the application, the result field will be automatically set to 0. When choosing any operation on valid inputs, the result of the operation will be displayed in the result field.

# 3. Design




My Model class is used for simplifying and checking the input string for any unaccepted characters. The actual model for the MVC pattern are the Operations and Polynomials classes, used by the Controller class, which is responsible with controlling the data flow into the model and updating the view whenever data changes. The Main class sets the view as visible and uses the Controller to act on the system.

**Packages:**

- GUI package: Contains the Model, View, Controller and Main classes
- Polynomial package: Contains the Monomial, Polynomial and Operations classes
- Util package: Contains the exceptions specific to the calculator: OperationException and WrongPolynomialException. Both exceptions extend RunTimeException.

## 4. Implementation

1) **Monomial**: Represents the monomials in the polynomial. It has two private field: coefficient (float) and exponent (integer).
   a) Monomial(float, int): Constructor method, which sets the coefficient and exponent of the Monomial object to the numbers given as parameters.
   b) hasEqualExponent(Monomial): Checks if the current Monomial object has exponent equal or not the the Monomial object given as parameter.
2) **Polynomial**: Represents the polynomial as an ArrayList of Monomial objects.
   a) Polynomial(): Constructor method with no parameters, which creates a new ArrayList of Monomials.
   b) Polynomial(List<String>): Constructor of polynomial with a linked list of strings given as parameter. Each string of the linked list contains a match group of the polynomial regex pattern. The coefficients and the exponents are extracted from the strings using the set pattern (ex. 2x^4) and new Monomials are created and added to the list.
   c) addMonomial(float, int): Adds a new Monomial to the list, using the Monomial's constructor.
   d) addMonomial(Monomial): Adds a new given Monomial to the polynomial.
   e) simplifyPolynomial(): Sorts the list of Monomial objects in descending order with respect to their exponent, then sums equal exponent elements and eliminates monomials with zero coefficient.
   f) findMonomialWithExponent(Monomial): Finds monomial in the list with exponent equal to the exponent of the monomial given as parameter.
   g) toString(): Constructs the string version of the polynomial. This will be of the form: 2x^5 + 3x^3 - 5x + 10 for example.
   h) degree(): Returns the degree of sorted polynomial, i.e. the largest exponent in the list.
   i) lead(): Return leading monomial (with highest exponent) of a sorted polynomial.
3) **Operations:** Implements static methods that do operations on one or two polynomials and return a result polynomial. The methods throw a new OperationException in case one of the operands is null.
   a) derivate(Polynomial): Method to calculate the derivative of a Polynomial. The derivative is calculated monomial by monomial and added to result Polynomial. The formulas used to calculate the derivative of a monomial are: newCoefficient = currentCoefficient * currentExponent and newExponent = currentExponent – 1.
   b) integration(Polynomial): Method to calculate the integral of a polynomial. The integral is calculated monomial by monomial and added to result polynomial. The formulas used to calculate the integral of a monomial are: newCoefficient = currentCoefficient / (currentExponent + 1) and newExponent = currentExponent + 1.

c) additionOrSubtraction(Polynomial, Polynomial, char): Method to calculate the addition or subtraction between two polynomials. The desired operation can be chosen with the op character: '+' for addition and '-' for subtraction. When choosing subtraction, the polynomial to be subtracted will have its coefficients multiplied by -1. After that, the addition is done monomial by monomial.

d) multiplication(Polynomial, Polynomial): Method to calculate the multiplication of two polynomials by multiplying each monomial with each one from the two polynomials (coefficients get multiplied and exponents get added).

e) division(Polynomial, Polynomial): Method to calculate division between two polynomials using long division algorithm. The method also checks for division by 0 after simplifying the two polynomials. If a division by 0 is to be performed, a new OperationException is thrown. It returns a list of two polynomials, first is the result of the division and second is the remainder.

4) **Model:** Responsible for the correct reading and validation of the input string given by the user.

a) checkPolynomial(String): Responsible with checking the input string for invalid characters, eliminating the spaces and '*' characters and splitting into monomials. This is done using regex and pattern matching. The regex used is: **([+-]?(?:(?:\\d*+x\\^\\d+)|(?:\\d+x)|(?:\\d+)|(?:x)))**. The checking of invalid characters is done using the replaceAll() method of the Matcher object. This method replaces all the input string's characters that are part of the regex pattern with "" (empty string). If the string at the end isn't equal to "", a new WrongPolynomialException is thrown.

5) **View:** Represents the visualization of the graphical interface.

a) resetAll(): Resets all the text fields to the initial value which is "0". It is used when the application is started or the Clear button is pressed.

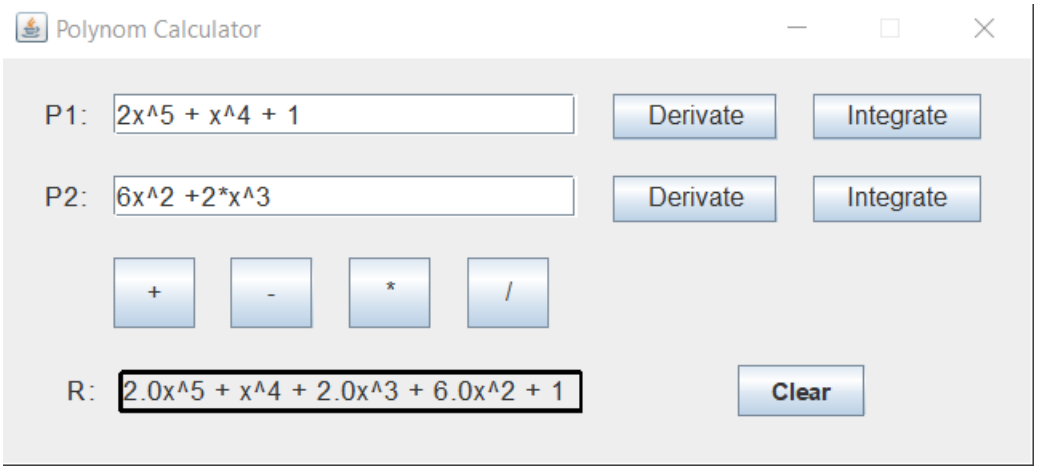
b) showError(Exception): In case of an exception, this method displays error message warning the user that something didn't go as planned.

c) addButtonListeners(ActionListener): Each button has such a method to add an ActionListener to it, so it can perform some task when it is pressed.

d) initComponents(): Sets up the graphical interface.

6) **Controller:** Controls the data flow into the Polynomial objects and updates the View whenever an operation is performed.

a) Controller(Model, View): Constructor method that sets up the Model, the View and the ActionListeners.



## 5. Results

To verify the correctness of the operations and methods used by the application, JUnit tests are created. They are grouped into packages and classes similar to the way that the application's classes are grouped. These test methods assert the actual return value of each method with an expected value.

In the polynomialTest package the Monomial, Polynomial and Operations classes are tested. The Monomial class only has one method tested, hasEqualExponent() to ensure the correct verification of equal exponents between two Monomial objects. The Polynomial class has its addMonomial(), simplifyPolynomial(), findMonomialWithExponent() and toString() methods tested. These methods are asserted with an expected value, created by using a LinkedList of strings containing monomials in the desired format and creating a Polynomial object using the LinkedList as a parameter to the constructor. Special cases are also checked, such as negative coefficients. The Operations class is tested using the same technique. Each operation is checked and special cases are taken into consideration.

The guiTest package contains a single class, modelTest. This class tests the checkPolynomial() method to ensure the correct validation and splitting of the input strings. Special cases are tested, for example negative coefficients or input strings given with spaces and '*' characters.

## 6. Conclusions

By following the secondary objectives, the created application achieves its main objective. The created calculator is simple and it could use some feature development, but it does the operations it should.

The strategy of using a list of Monomial objects to represent the Polynomial proved to be very useful in the implementation process. With this strategy, each operation on polynomials could be split into separate operations on each Monomial object in the Polynomial list.

The regular expressions and pattern matching features simplified the splitting and validation of input data. These features gave the user a lot more freedom as well. He/she doesn't have to follow a very strict format of entering the polynomial, for example it can be entered with or without spaces and '*' characters. This would have been much harder without the use of regex.

The exception handling part of the assignment made me learn a lot about such applications. I realized it is a big priority in the creation process to make the application stable and fix such exceptions so they don't crash the program. In the case of this Polynomial Calculator, exceptions could only happen in case of wrong input strings or operations on null polynomials.

JUnit testing proved to be very useful, because it made it possible to test code before being finishing the graphical interface of the application. Testing small parts of the program before giving the whole application a go saved me from a lot of headaches and it made it much easier. It is the first time I used this feature and will most definitely use it in my future Java projects.

Planning the project according to the Model View Controller architectural pattern made the code a lot more organized. Making changes to the graphical interface was a lot easier, as I only had to change the code in the View class.

In conclusion, this was a very interesting project that gave me the possibility of working with Object-oriented programming design. The assignment was challenging as I had to work with feature like JUnit and regex, which were new to me, but features which I will definitely use in my future projects because they proved to be very helpful.

For future development, I have 3 ideas:

1) Implementation of a more friendly and well-designed GUI, that also has a Help option where the user can get information about how to use the calculator.
2) Possibility to introduce negative exponents.
3) Possibility to introduce float coefficients.
4) An option to show the graphical representation of the entered polynomial as a graph.

# 7. Bibliography


- https://www.jetbrains.com/help/idea/tdd-with-intellij-idea.html
- https://en.wikipedia.org/wiki/Polynomial_long_division