Homework 1
Polynomials

Student:
Buzea Vlad-Calin
Group:30422

1.Objective

The objective of this project is to design and implement a system for polynomial processing. The polynomials should be of one variable and with integer coefficients.

2.Problem analysis

In mathematics, a polynomial is an expression consisting of variables, called indeterminates, and coefficients that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents. An example of a polynomial of a single indeterminate (or variable), x, is x2 − 4x + 7, which is a quadratic polynomial.

Polynomials appear in a wide variety of areas of mathematics and science. For example, they are used to form polynomial equations, which encode a wide range of problems, from elementary word problems to complicated problems in the sciences; they are used to define polynomial functions, which appear in settings ranging from basic chemistry and physics to economics and social science; they are used in calculus and numerical analysis to approximate other functions. In advanced mathematics, polynomials are used to construct polynomial rings and algebraic varieties, central concepts in algebra and algebraic geometry.

We will implement this problem from a general user point of view. That is with no different roles in the utilization. The application will be a simple calculator that works on polynomials.

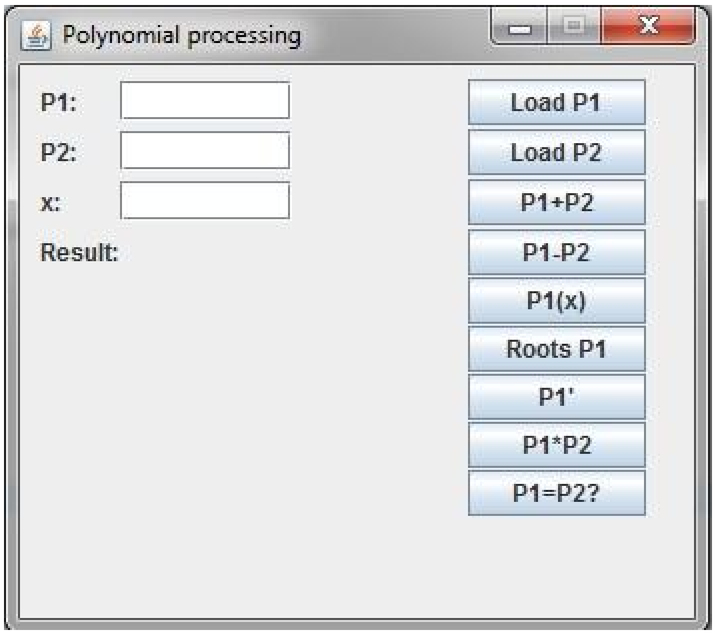
We will introduce two polynomials, P1(X) and P2(X), load them and perform the following operations:
-addition
-subtraction
-multiplication
-derivation
-finding integer roots
-calculation of the value of the polynomial in a specific point
-checking for equality of the two introduced polynomials

Optional operations:
-division
-integration
-graphic plot
These operations are impossible to do on integer coefficients, thus remaining for further development.

3. Application Development

User Interfaces

The proposed interface has the following appearance:



For now, we do not complicate the interface with menus and other windows. The first two text fields are used for introducing the polynomials in string form and the third is used for introducing the value of x, the point at which we want to calculate the value of the polynomial P1(X). The result of the computations will be displayed after the “Result” label. We can also see nine buttons on the right side of the window, each of them having a specific function:

- “Load P1” - evaluates the input and stores the correct values from the first text field
- “Load P2” - evaluates the input and stores the correct values from the second text field
- “P1+P2” – sums up polynomial P1 and polynomial P2
- “P1-P2” – subtracts polynomial P2 from polynomial P1
- “Roots P1” - finds and displays all the integer roots of the polynomial P1
- “ P1’ ” – differentiates the polynomial P1
- “P1*P2” – multiplies the two introduced polynomials
- “P1=P2?” – checks if the two introduced polynomials are equal

All the buttons, except “Load P1” and “Load P2”, produce an ephemeral result, that is displayed by the window and do not modify the stored values of the polynomials P1 and P2.

One can remark the appearance of guiding text, through labels

- “P1” precedes the input text box for the first polynomial
- “P2” precedes the input text box for the second polynomial
- “x” precedes the input text box for the value of the computation point
- “Result” precedes the computed result, also a text label.

Class Discovery

After an analysis of the requirements, 2 classes were created and used:

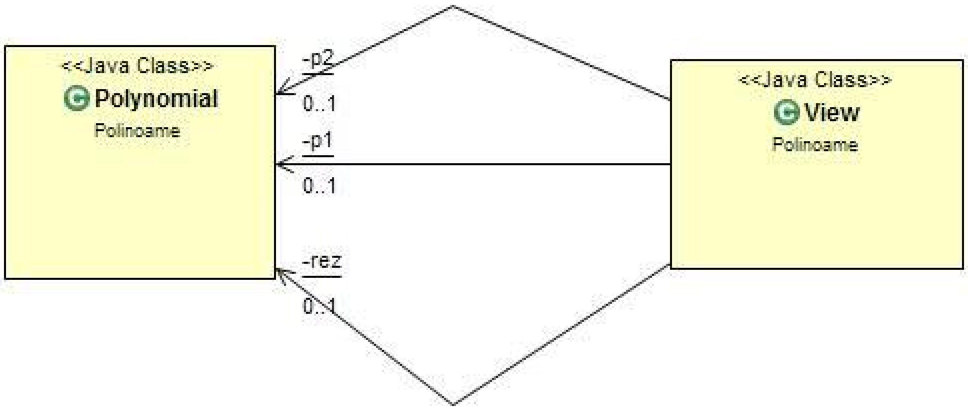
- **Polynomial:** it will be the mapping of the real world polynomial, containing the most relevant properties for the problem (the coefficients and the maximum degree) and the relevant operations
- **View:** contains the user interface elements and also correlates it with the corresponding functions implemented in the class Polynomial

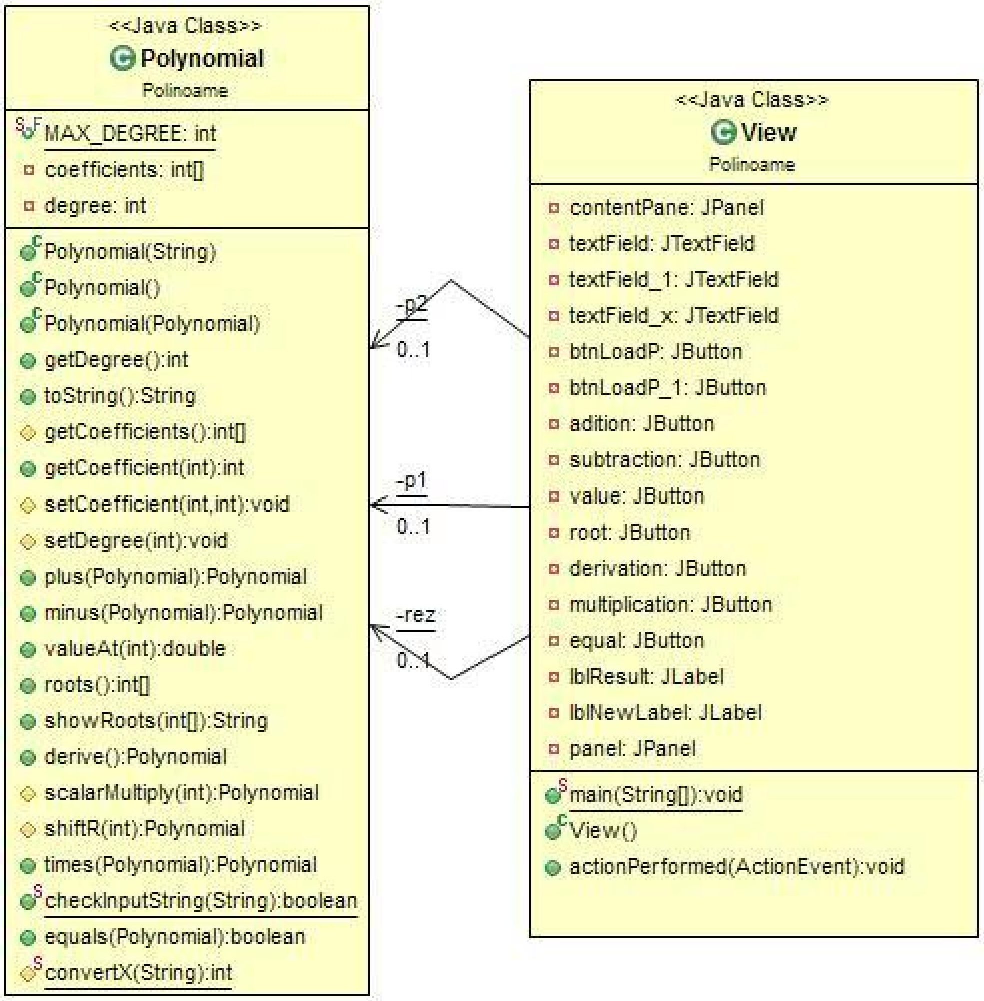
Further on the CRC cards are presented:

Polynomial	
- Saves the coefficients and the degree of a polynomial - Can compute the polynomial operations: addition, subtraction, multiplication, differentiation and evaluation (value at a given point x). - Finds the integer roots - Sets or returns coefficients from different degrees of the polynomial - Computes the degree of a polynomial - Sets or returns the degree of the polynomial - Converts the polynomial to a string representation - Finds the coefficients from a string representation of the polynomial - Compares for equality	

View	
- Handles the interaction of the program with the user - Represents the graphic interface - Has text fields for insertion of data - Executes the operations on polynomials - Loads polynomials and displays the results of the operations done on polynomials - Outputs results	- Polynomial

Class Diagram





It is clearly shown how the View class uses three instances of the Polynomial class.

4.Implementation and testing

The class **Polynomial** has two obvious fields: int[] coefficients and int degree. An array which stores the coefficients and the maximum degree of polynomial, also referred to as the maximum power with a non-zero coefficient associated. An alternative approach could have been the creation of the class Monome, that stores one coefficient and a corresponding power. The polynomial could have been formed out of several monomes. I have chosen the first implementation instead, due to the ease of computation and the resemblance to real world. As a constraint of this implementation, we must also have a class constant that indicates the maximum degree that any polynomial can have. We will name it **MAX_DEGREE.**

Class constructors are based on the transmission of the polynomial under String form. The String is then processed and the result is stored in fields. In order to obtain the wanted result, the polynomial must have the following form : "(coefficient)X^power_i+(coefficient)X^power_i-1+…+(coefficient)X+free_term" (example : 7X^5+(-1)X^4+(-X)+1). All coefficients must be preceded by the "+" sign in order to produce the wanted result, even if they are negative!

In order to verify the input String as little as possible, we implemented the static method " checkInputString" that returns false if the String doesn't respect the imposed constraints. These constraints are still under development, but the basic ones are:

-no other letter than x appears in the String (not case sensitive)
-x must be followed by ^ if not X^1 or the free term
-no other symbols are accepted except +, - , (,)

The process behind the constructor is based on splitting the String in processable chunks, using the "+" symbol. Thus we are left with 'coefficientX^power' pieces. So it is now easy to extract the data. Very useful for this operation is the split method, implemented the class String

(

public String[] split(String regex,

int limit)

Splits this string around matches of the given regular expression.

The array returned by this method contains each substring of this string that is terminated by another substring that matches the given expression or is terminated by the end of the string. The substrings in the array are in the order in which they occur in this string. If the expression does not match any part of the input then the resulting array has just one element, namely this string.

The limit parameter controls the number of times the pattern is applied and therefore affects the length of the resulting array. If the limit n is greater than zero then the pattern will be applied at most n - 1 times, the array's length will be no greater than n, and the array's last entry will contain all input beyond the last matched delimiter. If n is non-positive then the pattern will be applied as many times as possible and the array can have any length. If n is zero then the pattern will be applied as many times as possible, the array can have any length, and trailing empty strings will be discarded.

)

After the split, we extract the coefficient digit by digit until we get to ‘X’, jump the ‘^’ and compute the degree in the same manner. The extracted coefficient value is stored in the array at the position equal with the power. For eliminating case sensitivity (between ‘x’ and ‘X’), we apply the method toLowerCase, also implemented in the class String. For greater convenience we totally eliminate the ‘(‘ and ‘)’ characters from the transmitted String. This is because one may write “+(-5)X^3” and “+(-5X^3)” and still be correct. To do this we call the replace method from the class String.

Further on, we will present the implemented methods:

- `public String toString()`

Returns polynomial under String form: extracts value from the coefficients array appends ‘X^’ and the power index and continues process until it reached

X^1; here it only appends the coefficient and ‘X’; finally it appends the free term, situated at coefficients[0].

- `public int getDegree()`

It is a getter method in order to acquire the degree of a polynomial.

- `protected int[] getCoefficients()`

Also a getter method. Returns a copy of the array of coefficients.

- `public int getCoefficient(int position)`

Getter method for a specific coefficient, located at the transmitted position.

- `protected void setCoefficient(int position, int value)`

Setter method that modifies the values of the coefficients array in the specified position, introducing the transmitted value.

- `protected void setDegree(int value)`

Setter method for the degree. It should be used as a result of an operation that changed the real degree of the polynomial, thus making us have to set it.

- `public Polynomial plus(Polynomial p)`

This method is used to add the caller polynomial to the parameter polynomial and returns the result as a new polynomial. Addition is easy to implement because it doesn’t affect the degree of the polynomial. So, we just add the coefficient arrays, term by term (new_coefficients[i]=this.coefficients[i] + parameter.coefficients[i]).

- `public Polynomial minus(Polynomial p)`

The opposite operation of addition, subtraction has the same approach. The only difference is that we subtract the coefficients of the parameter polynomial p (new_coefficients[i]=this.coefficients[i] - parameter.coefficients[i]).

- `public double valueAt(int x)`

This method calculates the value of the polynomial in the point x, transmitted as parameter. The computation means to add coefficient*(x^i) for each monomer. This is the straight forward mathematical compution.

- `private int rootBound()`

Provides an integer upper bound for the value of the roots. It is a loose bound, but it is sufficient to stop our search early enough. We simply add all the modules of the coefficients, thus resulting our integer bound, which we return. (Source: http://en.wikipedia.org/wiki/Properties_of_polynomial_roots)

$$\max\left(1, \sum_{i=0}^{n-1}\left|\frac{a_i}{a_n}\right|\right)$$ (Hirst and Macey bound)[6]

- `public int[] roots()`

This method calculates the integer roots of the polynomial and returns them in a new array. The first element of the array (at position 0) is the number of found roots. The implementation is brutal, because it checks the values at all the points possible starting from 0 and ending at maximum and minimum integer value (at the same time). If the value of the polynomial at point i/-i is 0, we add i/-i to the array. Here is where we use the bound found with the rootBound() method, so that we only check for values smaller than it, not Integer.Max_Value.

- `public Polynomial derive()`

Differentiation of the polynomial is done by shifting the coefficients one position to the left, such the free term is lost. In the same time we must multiply the coefficient with the old power of the monome. Of course it is more convenient to start from coefficients[0], because we do not need any extra storage space. At the end of the procedure, the degree of the polynomial is decremented.

- `protected Polynomial scalarMultiply(int nr)`

Simply multiplies each element of the coefficients array with the parameter nr. Scalar multiplication, is included in the normal polynomial multiplication (multiplication with a polynomial of degree 0), but this particular method is used for the implementation of the normal multiplication method.

- `protected Polynomial shiftR(int positions)`

It does not represent a standalone operation on polynomials, but if we shift the coefficients array k positions to the right (towards the degree), we obtain the initial polynomial multiplied with X^k. This will be used in the regular multiplication method.

- `public Polynomial times(Polynomial p)`

Represents the final multiplication method, which includes the scalar multiplication and the X^k multiplication (shiftR()). The operation is done term by term:
-take one monome from the first polynomial
-multiply it with the second polynomial
-add the result to the already processed part
-repeat until we multiplied with each monome of the first polynomial
In other words, to multiply, for example 3X^4+1 with 2X^2+1 we multiply 3X^4 with 2X^2+1, store it in let's say aux, multiply 1 with 2X^2+1 and add it to the value of aux. The first part, 3X^4*(2X^2+1) is done by shifting the coefficients of the second polynomial with 4 positions and multiplying the modules with 3. Thus we obtain 3*(X^6+X^4).

- `public boolean equals(Polynomial p)`

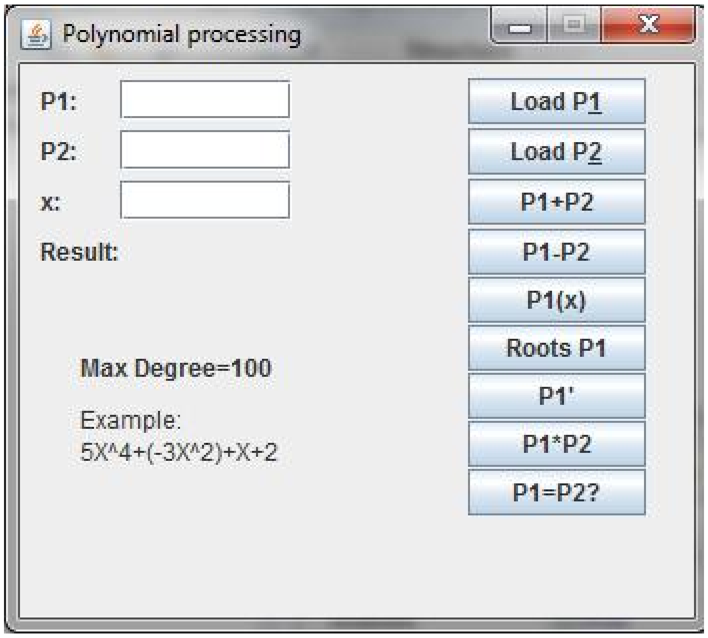
Verifies the equality of two polynomials by comparing first the degree, then the value of each coefficient. If one of the test is false, then the method returns false.

- `protected static int convertX(String s) throws Exception`

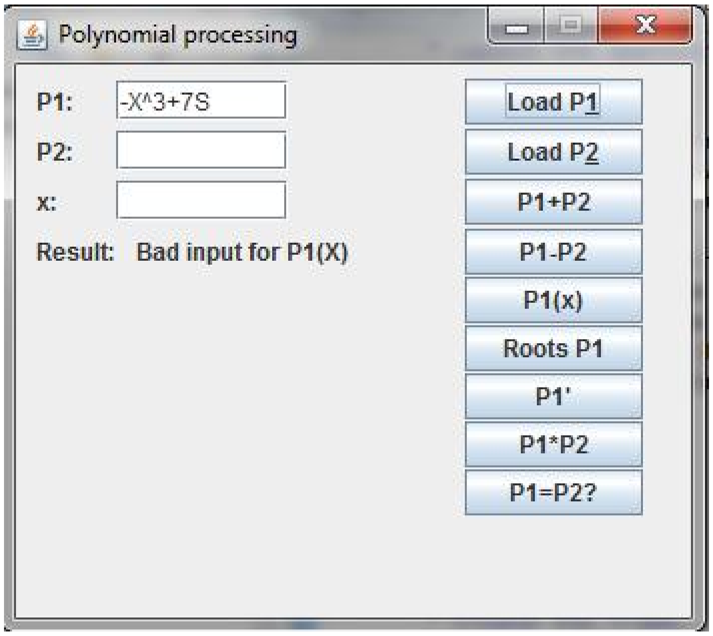
This is an inside method used to compute the value of the point x, transmitted under String form. It's main purpose is to convert the information transmitted in the text box into a manageable integer number.

The **View** is class in which we implement the user interface.

A help panel is set visible only when the cursor is inside the text box associated with P1. It shows the MAX_DEGREE and example of polynomial form.



When we press “Load P1” we create a new Polynomial with the String transmitted in the afferent text box. If the input does not correspond (checkInputString() returns false or the constructor sends an error) we show a message after the ‘Result’ label:



P1: -X^3+7S

Result: Bad input for P1(X)

If the operation succeeds we show the saved value after the ‘Result’ label and store the polynomial in a static variable p1. The validation is realized using try{}catch {} blocks, so that the program does not crash in case of a bad input. This also means that if a wrong value is introduced the old value of the polynomial is not discarded, thus making the application stay in good working condition.

The same sequence of steps is done for “Load P2”.

For the other buttons we simply call the corresponding functions implemented in the class Polynomial.

Mentionable is the way in which we detect the component on which the action was performed. We use the getSource() method from the ActionListener class and compare it to each button. This means all buttons are declared as class fields. If equality is met, then we just call the needed methods.

One may also remark few Help remarks given through the setToolTipText method.These are used on the buttons “Load P1” (Saves P1), “Load P2” (Saves P2) , “Roots P1” (Slow Computation) and on the text box corresponding to P1 (Insert P(X). All coefficients must start with + !!!)

5.Results

It is said that a picture is worth a thousand words. Well, then a video should be worth 24000 words a second.

As a demonstration of the obtained results (and also a testing method) I have inserted a video of the running application. If it is unavailable, please check the source folder for “Demo1.avi”.

Demo1.avi

One can clearly see the rejection of bad input data when other characters then X or x appear in the input. This is due to the single variable property of the polynomial. The aiding message is displayed on a panel made visible only when the cursor is hovering the area of the text box associated with P1. Aiding text is also displayed an some components, under the form of tool tip text.

I would also like to point out that the roots finding algorithm, although inefficient, stops very early in computation, thus the application doesn't remain blocked at any point of use. If we were to test all the possible values (until Integer.MAX_VALUE) the program would be stuck! Computing would take several minutes, time in which we wouldn't be able to do anything, not even close the application.

Besides the fact that all buttons respond as proposed, we can also see that the output doesn't change if a button is pressed many times. This is due to the fact that computations do not affect any of the transmitted polynomials! This can be changed if desired, but then a "debouncing" method should be considered. An undesired click may destroy the polynomial, or worse can lead to an unexpected final result. And an "Undo" button would also be wise in this case.

6.Conclusions

Polynomials have a great area of application and this program only scratches the surface. Computations regarding this subject are easy implementable once an working algorithm is devised. There are plenty of algorithms that target polynomials, so there is a lot of room for improvements.

From this homework I have learned how to manage many buttons with one ActionListener. Also, I have learned how to set a Help text near the mouse cursor when needed, using the setToolTipText() function. Another addition to my knowledge is the usage of WindowBuilder, an Eclipse IDE plug-in that allows drag and drop operations and visual aid in the development of the User Interface. Although such design is said to be volatile, aspect has a great deal of improvement. But the most important ability gained in this project was the to use the Eclipse IDE, a step long postponed until now.

Further development should reach the following points:

- Division of Polynomials
- Graphical representation of the function associated with the polynomial
- Integration of a Polynomial
- Fourier Series
- Laplace Transform
- Equations
- Composition : P2(P1(X))
- Complex roots
- Complex coefficients
- Efficient root computation
- Interpolation
- Bernstein Polynomials
- Full window resolution
- Introduction of constants (like Ⅲ, e $\ln 2, \log 2, \log_2 10$ and other such values)
- Computation of expanded coefficients
- Introduction of powers in a more elegant manner
- Menus: help, reset
- File input and output option
- Comparison of polynomials (show if P1≤ , ≥, <, > that P2)
- Expand number of polynomials, maybe even dynamically

7.Bibliography

- JDK Documentation
- http://en.wikipedia.org/wiki/Polynomial - Problem research
- http://en.wikipedia.org/wiki/Properties_of_polynomial_roots - Integer upper bound for the roots of the polynomial