Programming techniques

*Homework 1*

Moldovan Horia Andrei

Group 30424

Problem Specification :

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

Problem Analysis:

In order to design an application which fulfills the given specification we first have to understand the mathematical concept of the polynomial, what it represents, which are its components and the operations that can be done with them. In order to do that let us first start with the definition of a polynomial:

Def: A polynomial is an expression that can be build from constants and symbols called indeterminates or variables by means of addition, multiplication, and exponentiation to a non-negative integer power. Two such expressions that may be transformed, one top the other, by applying the usual properties of commutativity, associativity, and distributivity of addition and multiplication are considered as defining the same polynomial.{\displaystyle \sum \_{k=0}^{n}a\_{k}x^{k}}

A polynomial in a single indeterminate can always be written in the form:

P(x) = anxn + an-1xn-1 + … + a2x2 + a1x + a0,

Where an, …, a0 are constants and x is the indeterminate. The word “indeterminate” means that x represents no particular value, although any value may be substituted for it. The mapping that associates the result of this substitution to the substituted value is a function, called a polynomial function.

Decomposition of the complex problem in subproblems:

As presented above, a polynomial is nothing more than a collection of components of the form anxn. Each component of this form is called a monom. Furthermore, each monom has two components: coefficient and power. The power of the monom must be a non-negative integer and the coefficient can be any real number.

So in order to make our task easier we will define a polynom as a list of monoms, and we will make all the polynomial operations(addition, subtraction, multiplication, division, integration and differentiation) using the same operations but implemented on the monom.

--Monomial operations:

The operations which could be done on two monoms are :

1)Addition: we add the coefficients of the monoms iff the value of their power is the same.

2)Subtraction: we subtract the coefficient of the second monom from the coefficient of the first monom if the value of their power is the same.

3)Multiplication: we multiply the coefficients of the two monoms and add the value of their powers.

4)Division: we divide the coefficient of the first monom by the coefficient of the second monom and subtract the value of their powers.

The operations which could be done on one monom are:

1)Differentiation: we first multiply the value of the power by the value of the coefficient and then decrement the value of the power with 1.

2)Integration: we first increment the value of the power with 1 and then we divide the value of the coefficient with the incremented value of the power.

In each of this operations we create a new monom with the new computed coefficient and power.

After understanding the operations done on monoms we can proceed to the next step, which is the computation of the same operation on polynomials. In order to do that we will use the previously described methods.

--Polynomial operations:

The operations which could be done on two polynomials are:

1)Addition: we look in the two polynomials for monoms with the same power and use monom addition on them.

2)Subtraction: we look in the two polynomials for monoms with the same power and use monom subtraction on them.

3)Multiplication: we multiply each monom from the first polynom with each monom from the second polynom using polynom multiplication.

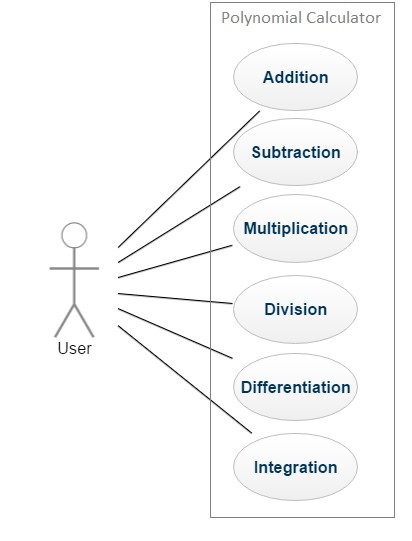
4)Division(the most complex operation of all): we first divide the first monom of the first polynom to the first monom of the polynom to be divided(second polynom) and the resulting monom will be multiplied with -1 and then multiplied again with the second monom and this will result in an auxiliary polynom. This new polynom will be added with the first polynom and the resulting polynom will technically be the new “first polynom”. We repeat this operation until the power of the first monom in the new “first polynom” will be lower than the power of the first monom in the polynom to be divided, the the “first polynom” will become the rest. We create a new polynom from all the monoms resulted from dividing by the first monom of the polynom to be divided and this polynom will be the quotient.

The operations which could be done on one polynomial are:

1)Differentiation: we use monom differentiation on each monom of the polynom.

2)Integration: we use monom integration on each monom of the polynom.

Use case

 The actor(user) is required to interact with the system by selecting an operation. From that point on, the system will decide if the inputs he provided are correct and will perform the necessary computations in order to provide a correct result to the user.

Use case: *Perform Addition of 2*  *polynomials* Primary actor: *User*

Main success scenario

* User launches application successfully.
* User provides first polynom by typing a string in the first text field provided by the application
* User provides second polynom by typing a string in the second text field provided by the application.
* User presses the “Addition” button from the given list of buttons.
* The first provided string is checked and successfully transformed into a polynom object.
* The second provided string is checked and successfully transformed into a polynom object.
* The addition is performed successfully with no exceptions or errors.
* The result is displayed in a label provided by the application.

Extensions

* If the user introduces a wrong input sequence in either of the two text fields, he will be noticed with a series of warnings and he can choose to retry introducing a correct and valid input sequence.

Use case: *Perform Differentiation of a polynomial* Primary actor: *User*

Main success scenario

* User launches application successfully.
* User provides a polynom in the first text field and he chooses if he wants or not to introduce a polynom in the second text field also but only what’s introduced in the first text field will be converted into a polynom.
* User hits the “Deriv” button.
* The provided string is successfully transformed into an object of type polynomial.
* The differentiation is performed successfully with no exceptions or errors.
* The result is displayed to the user.

Extensions

* If first polynomial is typed incorrectly, user will be noticed and will have the possibility to retype his polynomial without closing the application. As mentioned before, the second field is not taken into consideration when performing this operation.

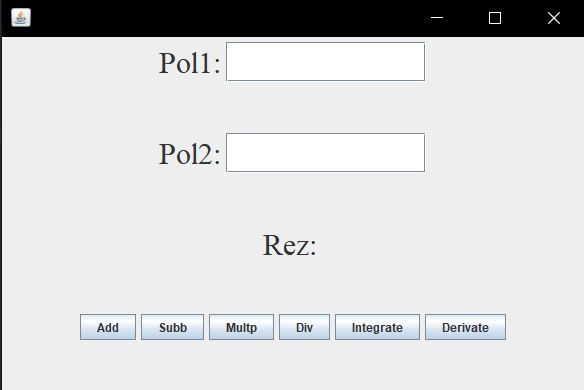
Design

We can divide the project in two major parts: the logic of the application and the user interface. They both must run flawless in order for the application to work correctly. The design follows a Model-View-Controller pattern.

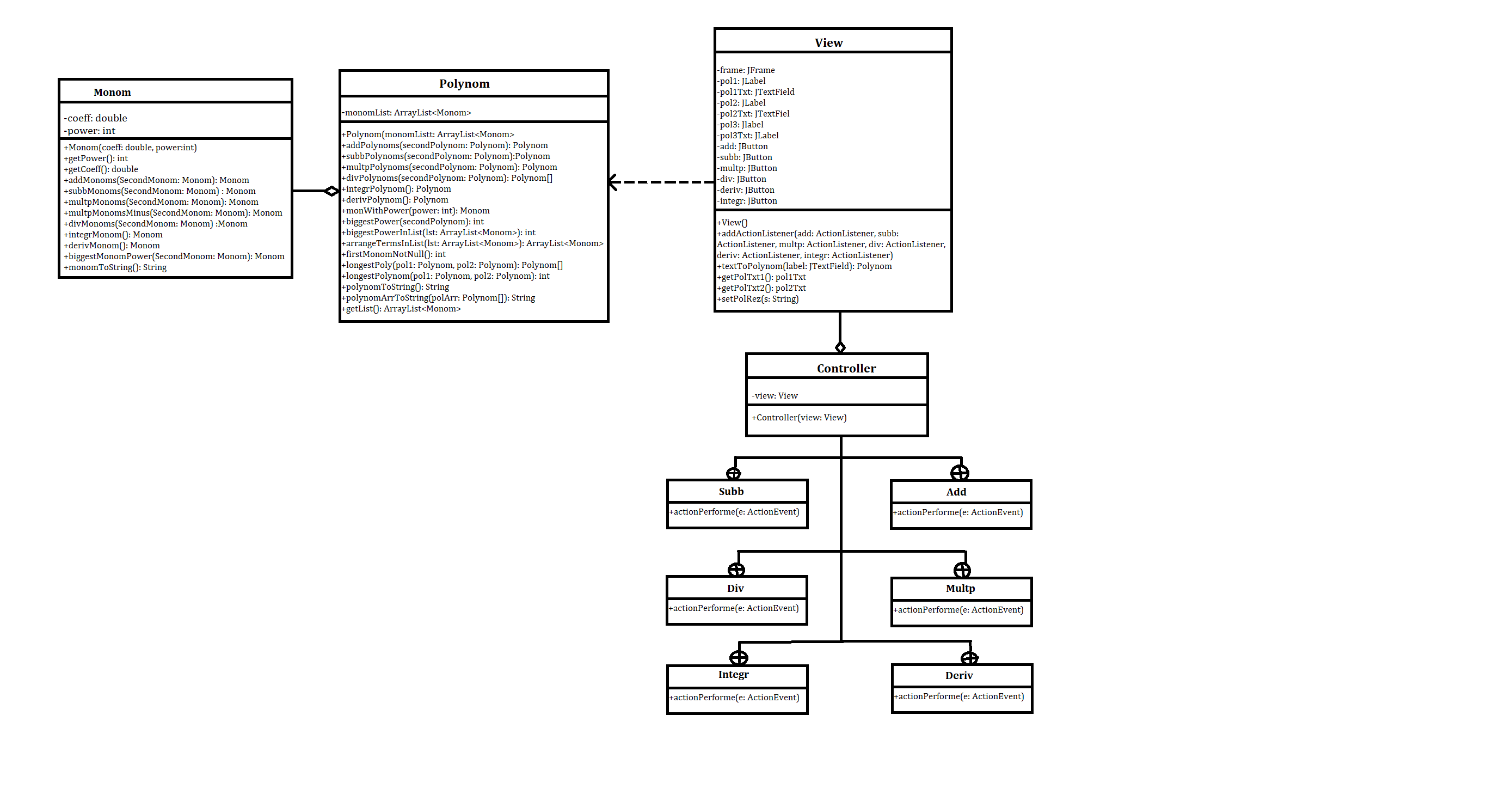
* The logic consists of the approach used(described above), and of the implementation(the classes with their attributes and methods), which glues all the parts of the MVC model together.
* The GUI part was implemented according to the MVC pattern by splitting the written code in three big parts. -The Model contains the algorithms which run “in the back”, to which the user has no access.

-The view is composed of two text fields, where the user should introduce the polynomials, a label which will show the result of the operations and six buttons which each correspond to one of the six operations which the user can perform on the introduced polynomials.

-The controller is composed of six action listeners, one for each button, where each one is used in order to make the connection between the view and the model, so in other words to link the button pressed by the user to the desired part in the code.



**UML Diagram**

****

**Implementation**

We will now take a closer look at the role and the implementation of each class with its attributes, constructors and methods.

* Monom.java

The attributes of this class are power, which is an integer and it represents the power of the monom and coeff, which is a double and represents the coefficient of the monom object. Both variables are declared private for security purposes and they can be accessed outside the Monom class only by using the getter methods getPower and getCoeff.

The constructor of this class takes as parameter two numbers of double and integer value and sets them as attributes to the monom object(as its coefficient respectively as its power)

The methods addMonoms, subbMonoms, multpMonoms, divMonoms, derivMonom, integrMonom are all methods used to compute the mathematical addition, subtraction, multiplication, division, differentiation and integration, which were all described above. We have and additionally addMonomsMinus method which will be used at the polynomial division. Not to forget that when adding or subtracting two polynomials there may be the case where in the second polynom there exists no monom with the power of some other monom from the first polynom, in this case I choose to add the monom with zero and a function which comes in help here is biggestMonomPower, because when adding a monom with zero we have to set the power to the resulting monom with the power of the initially monom, so this function tells us which was this power and sets it to the new created monom.

Another method is monomToString which is used to convert the computed monom into a type that can be used in order to show the result to the user, and this type is string. This method adds beside of the coefficient and of the power of the monom an additional “x”, representing the unknown variable and the sign “^” which is placed before the power of the monom. If the power is zero then only the coefficient will be converted into a string. So the resulting string will look like:

sign coefficient x ^ power , if the power is not equal to zero, or

sign coefficient , if the power equals zero.

* Polynom.java

The only field of this class is a list, monomList, which holds an array of the monoms constructing the polynom object. This field is declader private so it can be accessed outside the class only by using the getter function getMonomList

The constructor of this class takes as parameter a list of monoms and attributes it to the monom list of the polynom object when instantiated, so it creates a polynom object with an equation corresponding to the given list.

The function addPolynoms, subbPolynoms, multpPolynoms, divPolynoms, integrPolynoms, derivPolynoms are used in order to perform the requested mathematical operations on one or more polynom objects. This methods are using primary the methods created in the monom class but also methods created in this class, and we will see what each of them does when we encounter them in the mathematical methods.

The functions addPolynoms and subbPolynoms are very similary designed. They both take as parameter two polynoms and return a third one, the result. In the result we have to have as much monoms as the highest power of the monom with the highest power of the two added or subbed polynoms, and in order to find this number we use the function biggestPower which returns exactly what we need. Next we have to add or subtract monoms with equal powers, and as described above, when there is no monom with a specific power in one of the polynoms we said that we add or subtract with zero so in order to do that we need the function monWithPower which check is there exists a monom with a specific power in the monomList of the polynom and if there is nothing found the method creates a new monom which represents 0. Moreover we use the subbMonoms and addMonoms methods described in the monom class in oderd to perform our operation.

The method multpPolynoms takes two polynoms as parameter and return a new created polynom representing the multiplication of the two. Here we use the function longestPoly in order to determine which of the two polynoms have more monoms and this is a necessary information because otherwise we won’t know how to place our list iterators in the two for loops which make sure to multiply each element with each element using the described method addMonoms.

The methods integrPolynom and derivPolynom are similary made, the only difference being the mathematical concept of integration and differentiation applied on each monom of the polynom. So we go through the list of monoms and use the methods integrMonom and derivMonom at each iteration on the current monom.

The most complex method is derivPolynoms because the polynomial division is the hardest to compute, and in order to do it we have to make use of more other methods. This method takes as parameter two polynoms and it returns an array of polynoms which contain the quotient and the rest of the division. In order to successfully complete the division we have to follow the steps described above at the polynomial operations. We first declare some variables which will tell us throughout the operation on what place we are in the polynomials and when to stop the division. We follow each division step as described above. Some supplementary mentions would be that when we calculate the polynom to be subbed from the first one(auxToBeSubbed) we use the function multpMonomsMinus in order to multiply the result of the first multiplication by minus one. We repeat the same process as long as the grad of the rest is greater or equal with the grad of the quotient.

The functions polynomToString and polynomArrToString simply use the function monomToString in order to convert the list of monoms to text which can be send as output.

* View.java

This is the class which builds the aspect of the user interface. The components used were described above at the design part. This class has two methods. One is used as action listeners for the buttons which are related to the mathematical operations, buttons which are implemented in the controller. The other one is used in order to convert the string from the text labels into polynomial objects. In order to do that I let the introduced monom to be of the previously described fixed form which is sign , coefficient, x, ^, power. This form is used by this converting method because it creates monoms by taking the values of the second and fifth position of the monom string and puts them into the new created monom. It returns a list of all the introduced monoms so basically a polynom.

* Controller.java

In this class is done the linking between the buttons created in the view class and the mathematical operations created in the polynom class. In order to do that, the extracted string from the text fields is converted into polynoms and then the needed operation is performed on them and finally the resulted polynom is converted back to a string and set as result in the view declared label for the result in so that the user can see it as well.

* JUnitTest.java

This is the class which was used to test the methods defined in the other classes. Using the function assertEquals we could see if the expected result is the same with the actual result.