Student: Ardelean Eugen-Richard

Group: 30423

Programming Techniques

Homework 1

Contents:

1. Problem Specification
2. Analyzing the problem, modelling, scenarios, use cases
3. Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)
4. Implementation and testing
5. Results
6. Conclusion and future developments
7. Bibliography
8. **Problem Specification**

This first assignments consists of proposing, designing and implementing a system for polynomial processing. We should take into consideration also the fact that these polynomials are of one variable and have integer coefficients.

There also are some mandatory requirements in designing this project: using encapsulation, the classes have a maximum of 300 lines of code (exceptions the GUI classes) while the methods a maximum of 30 lines and remember to use the java naming conventions.

Mandatory for passing are also the use of the graphic interface, the implementation of addition and subtraction and the documentation. For extra points, the following operations can be added to the application: multiplication, division, derivation and integration. Also, there are extra points given for designing in an object-oriented manner and testing the project’s most important parts using Junit tests.

The graphical interface will allow the entering of data, the 2 polynomials, the selection of the desired operation and then it will output the result of the operation in the same form as the input.

1. **Analyzing the problem, modelling, scenarios, use cases**
   1. **Problem analysis**

A system that can process polynomials is useful for a wide range of users, from students who need help with their exercises to experts that need a system that can calculate complex operations. The graphic interface is easier to use than the console by users that do not have an interest in computer science.

In mathematics, a monomial is, technically speaking, a polynomial with one term. It is also called a power product because it’s a product of powers of variables with non-negative integer, multiplied by a non-zero constant, called the coefficient of the monomial. Whereas, the polynomial is an expression that contains constants, variables and exponents. The variable’s exponents can be only non-negative integers, just as in the monomial, and it must have a non-infinite number of terms(monomials).

Each operation has a certain mathematic algorithm to get the result. For adding and subtracting 2 polynomials you can only add the monomials that have the same degree, the operation is done on the coefficients and the elements that don’t have the same degree in both polynomials are added with the appropriate sign. The multiplication is done in the following manner, each monomial from the first monomial is multiplied with each of the monomials from the second one, then the monomials of the result that have the same degree will be added (as presented in the addition). The division of 2 polynomials consists of multiple steps: you must divide the first term of the dividend by the highest term of the divisor then you must multiply the divisor by the result just obtained after that subtract the product just obtained from the appropriate terms of the original dividend and repeat the previous three steps, except this time use the n terms that have been written as the dividend. The derivation and integration are both done by deriving or integrating each monomial of the polynomial.

* 1. **Modelling**

Because we know that the system has one variable and integer coefficients, we can build the polynomials using 2 classes: monomial and polynomial.

The monomial and polynomial classes have been modelled to represent their mathematical equivalents. Just as in mathematics, a polynomial is built from one or more parts, each having a degree and a coefficient, called a monomial that is separated from the other monomials by a plus or minus sign. For both classes monomial and polynomial there will be created operations, that represent the operations you can do with the monomial or polynomial. The addition of monomials will be different from the one of polynomials, but the polynomial one will use the monomial addition.

* 1. **Scenarios**

The application allows the user to enter 2 polynomials and choose an operation that the user would like to be executed.

For example, if we have the followings polynomials as inputs:

Polynomial 1 as p1 for simplicity: p1 = 5x^2 - 3x + 1

Polynomial 2 as p2: p2 = x^2 + 4

For the following operations, the results will be:

Addition result: p1 + p2 = 6x^2 - 3x + 5

Subtraction result: p1 - p2 = 4x^2 - 3x - 3 and p2 - p1 = -4x^2 + 3x - 3

Multiplication result: p1 \* p2 = 5x^4 - 3x^3 + 21x^2 - 12x + 4

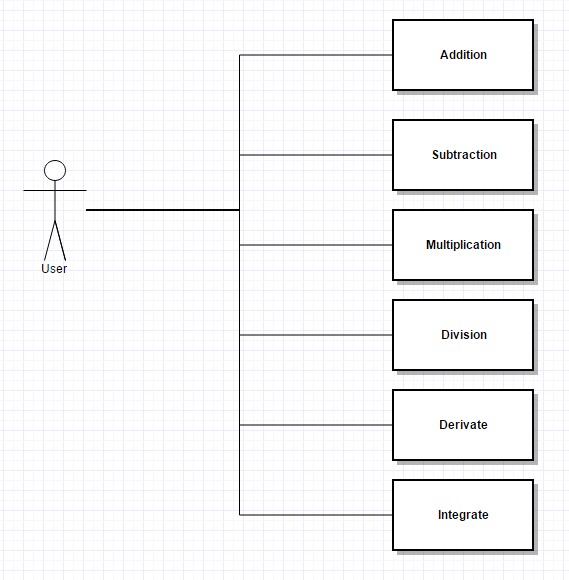
Division result: p1 / p2 = quotient of 5 with a remainder of -3x - 19

Differentiation result: p1’ = 10x - 3 and p2’ = 2x

Integration result: and

* 1. **Use cases**

A use case diagram represents the user interacting with the application.



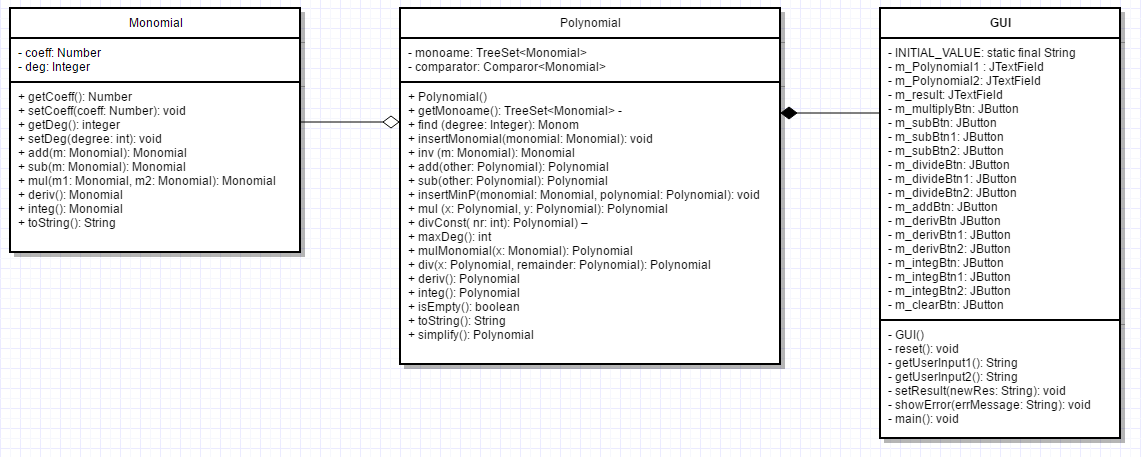
1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)**
   1. **Design decisions**

I chose to use the object-oriented approach with a class for monomials and one for polynomials. In the monomials class, for each operation there is a method that represents how that operation would work between 2 monomials. Coefficient and degree are the only instance variables used in the class. Just as in the monomial class, in the polynomial class we have a method for each operation that calculates the result between 2 polynomials. To represent a polynomial, a TreeSet<Monomial> was used instead of a List<> because it sorts the data that is introduced. For convenience, most of the methods (that represent operations) have been created to use the current object as one of polynomials.

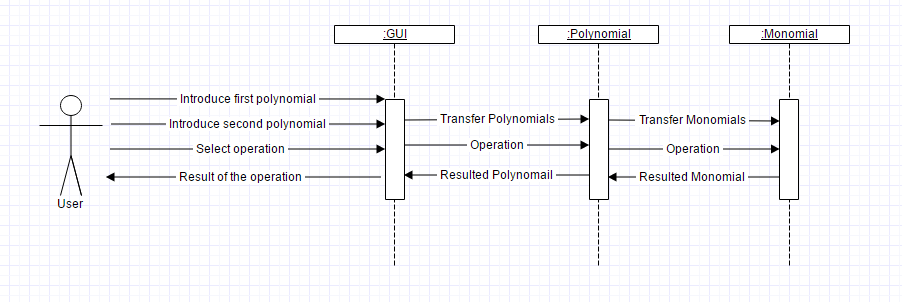
* 1. **UML diagrams**

We prepare UML diagrams to understand a system in better and simple way. A single diagram is not enough to cover all aspects of the system. So, UML defines various kinds of diagrams to cover most of the aspects of a system.

Class diagrams are the most common diagrams used in UML. Class diagram consists of classes, interfaces, associations and collaboration. Class diagrams basically represent the object-oriented view of a system which is static in nature.



A sequence diagram is a diagram that shows the interaction between the different objects of the application. For Polynomial Processing it would look something like that:



* 1. **Relationships**

In the current version of the application there are no inheritance relationships between classes.

But relationships of aggregation are present between classes: *Monomial* and *Polynomial,* the relationship is of aggregation type because technically objects of type Monomial can exist without object of type Polynomial.

Between the polynomial class and graphical user interface one there is a relation of composition because the user interface cannot exist without polynomials.

* 1. **Class Design**

For this application, I used 2 main classes for the implementation, one for monomials and one for polynomials.

In the monomial class:

* Attributes: the normal attributes for a monomial, the coefficient and the degree
* Constructors: there are 2 constructors, the first has no parameters and it sets the monomial to coefficient 0 and grade 0 and the second has 2 parameters, one an integer and the other of type Number, that are used to initialize the monomial
* Methods:
  + getCoeff(): Number – when called, returns the coefficient of the monomial
  + setCoeff(coeff: Number): void – when called sets the coefficient of the monomial to that of the parameter
  + getDeg(): integer – when called, returns the degree of the monomial
  + setDeg(degree: int): void – when called sets the degree of the monomial to that of the parameter
  + add(m: Monomial): Monomial – when called adds the parameter to the current object
  + sub(m: Monomial): Monomial - when called subtracts the parameter from the current object
  + mul(m1: Monomial, m2: Monomial): Monomial – when called multiplies m1 with m2 and puts the result in the current object
  + deriv(): Monomial – when called derivates the current object
  + integ(): Monomial – when called integrates the current object
  + toString(): String – an overridden method used to return the classic appearance of a monomial (ex: 2x^3)

In the polynomial class:

* Attributes: there is a TreeSet that represents the polynomial (I chose TreeSet because it is sorted) and a Comparator which is used in the constructor to be able to write it from the highest to the lowest degree, instead of the normal sorting of the TreeSet from lowest to highest.
* Constructors: one constructor with no parameters that initializes the polynomial
* Methods:
  + getMonoame(): TreeSet<Monomial> - returns the TreeSet monoame
  + find (degree: Integer): Monomial – returns the monomial if it finds a monomial with the same degree as the parameter in the polynomial, otherwise it returns null
  + insertMonomial (monomial: Monomial): void – inserts the monomial in the parameter in the polynomial
  + inv (m: Monomial): Monomial – inverts the coefficient, it is used for negative values (ex: in subtraction)
  + add(other: Polynomial): Polynomial – adds the polynomial in the parameter to the current object(also a polynomial)
  + sub(other: Polynomial): Polynomial – subtracts the polynomial in the parameter from the current object
  + insertMinP(monomial: Monomial, polynomial: Polynomial): void – inserts the monom from the parameter in the polynomial from the parameters
  + mul (x: Polynomial, y: Polynomial): Polynomial – multiplies the polynomials x and y and puts the result in the current object’
  + divConst( nr: int): Polynomial) – divides the polynomial with the integer in the parameter
  + maxDeg(): int – finds the maximum degree of a monomial in the polynomial
  + mulMonomial(x: Monomial): Polynomial – multiplies the current object with the monomial in the parameter
  + div(x: Polynomial): Polynomial – divides the current object to the parameter polynomial
  + deriv(): Polynomial – derivates the current object
  + integ(): Polynomial – integrates the current object
  + isEmpty(): boolean – verifies if the current object has no monomials
  + toString(): String – an overridden method that is used to display the classic form of a polynomial
  + simplify(): Polynomial – removes the elements in the polynomial with coefficient 0

There also is also a GUI class that uses ActionListeners to do the operations when clicking a button.

In this class, I used Swing elements (textfields, buttons, labels, panels) to represent a user-friendly interface for polynomial processing.

* 1. **Algorithms**

There are many algorithms worth mentioning in this section, but the most representative would be the polynomial division.

The pseudocode algorithm for the polynomial division is the following one:

quotient ← 0

remainder ← n # At each step n = divisor × quotient + remainder

while remainder ≠ 0 AND degree(remainder) ≥ degree(divisor):

temp ← maxDegree(remainder) / maxDegree(divisor) # Divide the leading terms

quotient ← quotient + temp

remainder ← remainder − temp \* divisor

return (quotient, remainder)

* 1. **Data structures**

The only data structure used in the implementation is the TreeSet<Monomial>, containing elements of monomial type. To order the Tree Set, a Comparator was used. The comparator is a function with parameters a and b, both monomials, and it compares the degrees of the monomials using the function compareTo and according to what compareTo returns it orders the TreeSet.

**private** TreeSet<Monomial> monoame;

**private** Comparator<Monomial> comparator = (Monomial a, Monomial b) -> a.getDeg().compareTo(b.getDeg());

Because of this implementation, the limit of the degree of a polynomial is the limit of the data type.

* 1. **Packages**

For the current implementation, all classes have been saved in one package. The GUI class and the monomial and polynomial classes are easier to understand if they are all in one package, because they are tightly connected between each other.

* 1. **Interfaces**

For the implementation of this project, there were no interfaces used as there wasn’t any need to implement them. All classes have different methods, even if they have the same name, the way they operate and the parameters are different.

* 1. **User interface**

The User Interface was built using Java Swing elements (buttons, panels, frames, labels, textfields).

A window will open when running the application. This window will allow the user to introduce 2 polynomials as inputs in 2 appropriately labeled textfields, and then choose an operation by clicking one of the appropriately named buttons.

For executing the selected operation, the predefined ActionListener interface is used. This allows us to make changes when an action is performed, like clicking a button. After choosing the operation, by clicking the desired button, the result will be displayed in a non-modifiable textfield labeled “Result:”.

For the following operations: subtraction, division, derivation and integration there are choices, that have been implemented as another 2 buttons that show you what choices you have, for examples for subtraction they will appear labeled as “p1-p2” and “p2-p1”, each doing exactly what it says.

For subtraction, you can choose to subtract the first polynomial from the second or the second one from the first.

For subtraction, you can choose to subtract the first polynomial from the second or the second one from the first.

For division, you can choose to divide the first polynomial from the second or the second one from the first.

For derivation, you can choose to derivate the first or the second polynomial.

For integration, you can choose to derivate the first or the second polynomial.

A clear button was also created, it will allow you to clear all the textfields, both inputs and outputs and it will reset the result to 0. The clear button will also “reset” the buttons, showing the operation names and not the choices.

1. **Implementation and testing**
   1. **Implementation**

The polynomials are introduced as a sum of monomials. The inserted polynomial must have the following appearance:

sign + coefficient + “x^” + power + ”+”

* Sign: “-“ for negative and nothing for positive
* Coefficient: must be an integer number
* Power has to be an integer value

Examples of valid inputs:

1x^2+-3x^1

-5x^3+13x^0

4x^0+3x^2

The monomials do not have to be entered in decreasing order of the degrees but all powers and coefficients should be written including degree 0 and degree 1. The only supported characters are numbers, the variable “x”, the power sign “^”, the operator “+” and the sign “-“.

The best example of implementation algorithm is the division, it looks like this:

**public** Polynomial div(Polynomial x, Polynomial remainder) //polynomial division function

{

Polynomial auxP = **new** Polynomial(); // auxiliary first Polynomial

Polynomial auxD = **new** Polynomial(); // auxiliary second Polynomial

Polynomial prevD;

**for**(Monomial MonomialX: **this**.getMonoame())

{

insertMinP(MonomialX, auxD); //copy the values from polynomial(current object, dividend) into auxD

}

**if**(x.maxDeg() == 0) //if the second polynomial has the highest degree 0 => second polynomial is a constant

**for**(Monomial m: **this**.getMonoame())

**this**.divConst(m.getDeg()); // so you divide each coefficient of the first polynomial to the constant

**else**

{

**while**(auxD.maxDeg() >= x.maxDeg() && x.find(x.maxDeg()).getCoeff().floatValue() != 0) // if the divisor is bigger than the dividend then dividend = remainder so it goes out of the loop

{

**float** coefD = auxD.find(auxD.maxDeg()).getCoeff().floatValue() / x.find(x.maxDeg()).getCoeff().floatValue(); //for each division create a coefficient

**int** degD = auxD.maxDeg() - x.maxDeg(); //for each division create a degree

Monomial m = **new** Monomial(coefD, degD); // the new coefficient and degree create a Monomial

auxP.insertMonomial(m); //this Monomial is inserted in auxP

prevD = **new** Polynomial(); // new polynomial

**for**(Monomial MonomialX: x.getMonoame())

{

insertMinP(MonomialX, prevD); //initialize the new polynomial with the values of polynomial x by inserting in prevD the monomial

}

prevD = x.mulMonomial(m); // multiply prevD with the monomial created before

auxD.sub(prevD); // subtract from auxD, prevD

}

**this**.monoame = auxP.monoame; //copy auxP-quotient into the current object

**if**(auxD.maxDeg() > 0 || auxD.find(auxD.maxDeg()).getCoeff().floatValue() != 0)

{

remainder.monoame = auxD.monoame; //copy auxD-remainder into the remainder polynomial

}

}

**return** **this**;

}

* 1. **Testing**

The testing of the implementation was done both by the GUI interface and the use of Junit tests. A class called PolynomialTests was created in which a test method for each operation was made.

For each operation, there were created the appropriate number of polynomials for the test.

For example, for addition there were created 3 polynomials, 2 for operands and one for the result.

In the operands, random monomials where introduced and in the result, the result of the addition of the 2 polynomials. Then the addition function was called and after that the assertEquals function to compare the result of the addition and the result polynomial.

@Test

**public** **void** testAdd() {

Polynomial a=**new** Polynomial();

Polynomial b=**new** Polynomial();

Polynomial c=**new** Polynomial();

a.insertMonomial(**new** Monomial(1,1)); //inserting a random monomial in a

a.insertMonomial(**new** Monomial(2,2)); //inserting a random monomial in a

b.insertMonomial(**new** Monomial(1,2)); //inserting a random monomial in b

c.insertMonomial(**new** Monomial(3,2)); //inserting a monomial of the result in c

c.insertMonomial(**new** Monomial(1,1)); //inserting a monomial of the result in c

a.add(b); // adds the 2 polynomials and puts the result in a

*assertEquals*(a,c); //compare a and c

}

For assertEquals to work, a method equals has to be created in class Polynomial:

**public** **boolean** equals(Object a) // we also override the equals methods

{

**int** ok=1;

Polynomial b=**new** Polynomial();

//copy this into b

**for**(Monomial bla: **this**.getMonoame())

b.insertMonomial(bla);

//subtract a from b and if you get 0 then its equal

b.sub((Polynomial)(a));

//sees if all coefficients are 0

**for**(Monomial bla: b.getMonoame())

**if**(bla.getCoeff().floatValue()!=0)

ok=0; //signals if it has any coefficient different from 0

**if**(ok==1)

**return** **true**;

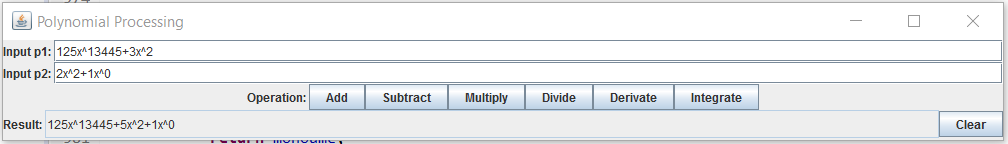
**return** **false**;

}

1. **Results**

Through hard work and intensive testing, I have been able to create an application for Polynomial Processing. The application has all operations fully functional.

The application looks like this:



1. **Conclusions and future developments**
   1. **Conclusions**

Implementing a program like this can be hard when deciding what to use for the implementation and how to implement it (ex. To have 2 polynomials and update one with the result or have 3 polynomials and put the result of the operation in the third). Implementing the multiplication and the division was especially difficult even after managing to implement the others. The number of hours was more than expected in solving this project. The number of hours may be because of the lack of exercise with programming techniques and because of the fact that I had to rethink the algorithms 5-10 times until I got them right. Deciding for an efficient way to have float coefficients for the division and integration operations was quite difficult, but the implementation of it was easier than expected.

* 1. **Future developments**

There are many additions that can be made to this application, like improving the range of functionality.

Examples of improvements:

* Coefficients of float type for real coefficients
* Doing operations with polynomials that have more than one variable
* There can be more operations, such as: finding the root of a polynomial, calculating the polynomial for a certain value, verify equality which could be used in case of long unordered polynomials
* The input has a strict form; it could be loosened to make it easier for users to introduce data
* There could be implemented a button that can switch between the inputs instead of having a button for deriving the first polynomial and another for deriving the second one
* There could be implemented a button that can put the output in one of the inputs of our choice, for n-grade derivation or just implement the n-grade derivation
* Draw the graphic for the polynomial functions

1. **Bibliography**

<http://stackoverflow.com/>

<http://docs.oracle.com/javase/7/docs/api/>

<https://www.gliffy.com/>