TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Programming Techniques - Assignment 1

Polynomial Processing

Student: Tosa Bianca Ioana

Group: 30422

Contents

1. Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2. Problem analysis, modeling, scenarios and use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.1. Problem Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.2. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3. Scenarios and use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3. Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1. UML Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1.1. GUI Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1.2. Term Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1.3. Polynomial Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1.4. Termscomparator Class . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1.5. BiOperation and UniOperation Interfaces . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1.6. Classes that implement BiOperation Interface . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1.7. Classes that implement UniOperation Interface . . . . . . . . . . . . . . . . . . . . . . . 7

3.2. Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

3.3. Relationships . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.4. Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.5. User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4. Implementation and Testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

6. Conclusion and Feature Developments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

7. References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .12

1. Problem specification

Assignment 1 - Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

The program will be able to perform the following operations on the polynomials given by the user:

* Addition of two polynomials
* Subtraction of two polynomials
* Multiplication of two polynomials
* Division of two polynomials
* Derivation of one polynomial
* Integration of one polynomial

The program will have a graphical user interface through which the user will be able to insert the polynomials and choose which of the given operations would like to apply. After the introduction of polynomials the user will press the “Done” button and will see on screen the given polynomials. The monomials of each polynomial will be displayed in descending order of their degrees. The result of the chosen operation will be displayed on the screen, in the bottom-right part of the panel.

1. Problem analysis, modeling, scenarios and use cases

2.1. Problem Analysis

The most efficient way to store a polynomial and also keep a proper OOP style of the design is to split it into monomials, each of them having a coefficient and a degree. Using this approach, in order to add or subtract two polynomials the program has to find the monomials with equal degrees and add or subtract their coefficients. The multiplication is done in the same way, by calculating the product of any two terms and then add the coefficients of those terms in the result that have the same degree. For division, one can use the Euclidean algorithm, which will be easy to implement due to the simple access of the coefficient and degree of the terms. The integration can be done by increasing the degree of each term with 1 and divide the coefficient with the new power. In the same way, after derivation, the new coefficient of a monomial will be the old one multiplied with the degree and the new power will be decreased by 1.

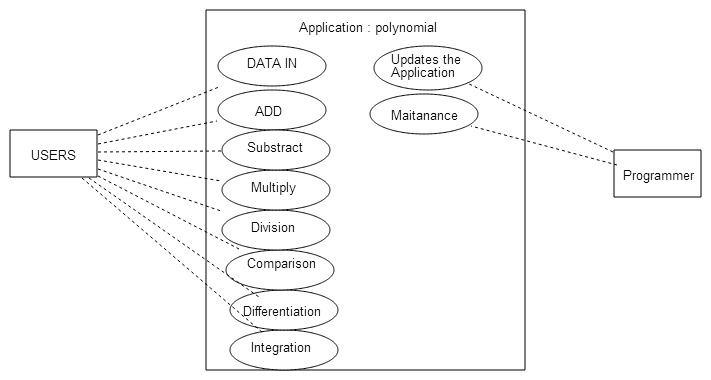
2.2. Modeling

Based on the ideas presented in section 2.1 the implementation of this project will be done in an Object Oriented Programming style. In order to keep a good organization of the necessary classes, I grouped them in 4 packages: Main, Interface, Operations, PolynomInfo. The Main package contains only the Main class through which the entire program is run. GUI class is responsible for the graphic user interface and is held in the Interface package. The third package mentioned above contains, as the name suggests, a class for each of the requested operations. Moreover, in order to maintain the desirable OOP style, Operations package has two interfaces, which correspond to operations made on one or two polynomials. The last package contains three classes: Term class stores the degree and the coefficient of each monomial. The monomials are stored in an ArrayList, declared in the Polynomial class, which also contains methods for sorting and printing a polynomial. The last class in this package is TermsComparator, which is used for sorting the ArrayList of Terms by their degrees.

2.3. Scenarios and Use Cases

The user has to enter each polynomial in a JTextField, labeled “First polynomial” and “Second polynomial”, term by term, in this way: “coefficient x^ degree”, although the degree or the coefficient is 0 or 1. However, after pressing the “Done” button, the result will be displayed on the screen without the undesired coefficients or degrees of 1 and 0. After introducing the polynomials, the user can choose which operation will like to perform. Integration and derivation will be performed on both polynomials and the result will be labeled properly.

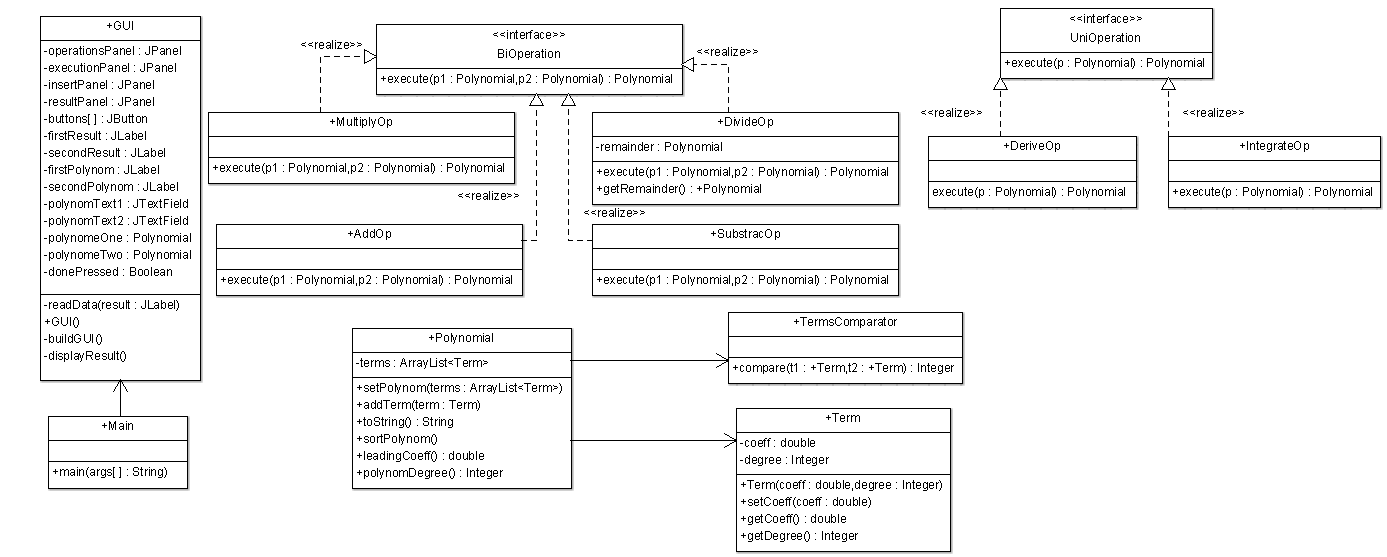
The graphical interface is designed as user-friendly as possible, such that it cannot provide any problems of understanding what the application does.

Below, there are presented the use cases, which as shown in the diagram are strongly related to the user. 

1. Design

3.1. UML Diagram

The UML diagram shows the relationships between classes, the attributes and methods in all classes and also, their type: public or private.



3.1.1. GUI Class

This class has three methods :

* readData() – takes the polynomials introduced into text fields by the user and transforms them into ArrayLists of Terms;
* buildGUI() – constructs the user-friendly interface, with all the buttons and labels that help the user to understand the program;
* displayResult() – prints on the screen a label of the requested operation followed by the result of it.

3.1.2. Term Class

Term class has two attributes corresponding to the coefficient and the degree of a monomial. It has a constructor and also setters and getters for the two attributes.

3.1.3. Polynomial Class

This class contains an ArrayList of Terms and all the methods that involve a polynomial:

* setPolynom() – this method is the constructor of the Polynomial class;
* addTerm() – it is used to add a new Term to the ArrayList, avoiding to have two terms with the same degree;
* toString() – is an overridden method used to display a polynomial in an accurate way
* sortPolynom() – this method sort the monomials composing a polynomial by their degrees
* leadingCoeff() – gives the coefficient of the term with the highest degree; this method is useful while applying the Euclidean algorithm for the division of two polynomials;
* polynomDegree() – returns the degree of the polynomial, which is the maximum degree of all terms.

3.1.4. TermsComparator Class

This class implements the Comparator and contains only one method, which is used for sorting the terms in a polynomial.

3.1.5. BiOperation and UniOperation Interfaces

These two interfaces have the purpose to maintain a proper OOP style. Each interface has one method, one with two parameters of type Polynomial, and the other one with only one parameter of the same type. As shown in the UML diagram, classes AddOp, SubtractOp, DivideOp and MultiplyOp implement BiOperation interface, because all these operations are made on thw polynomials. Using the same logic, it is easy to understand why classes IntegrateOp and DeriveOp implement the UniOperation interface.

3.1.6. Classes that implement BiOperation

* AddOp Class – in this class is performed the addition of two polynomials. In order to be more efficient, it uses the addTerm(Term term) method from Polynomial class, so that the result will not contain two or more terms with same degree.
* SubtractOp Class – similarly to the previous class, this one is responsible for subtracting two polynomials;
* DivideOp Class – the division of two polynomials was made based on the Euclidean alghoritm. The result is displayed as two distinct polynomials, a quotient and a remainder, both of type Polynomial.
* MultiplyOp Class – this class implements its method to give the product of two polynomials

3.1.7. Classes that implement UniOperation

* DeriveOp Class–this class implements the execute(Polynomial p) method in order to return the given polynomial differentiated;
* IntegrateOp Class – the method in this class does the opposite of what the one in the DeriveOp class does – it returns the integration of the given polynomial.

3.2. Data structures

As mentioned before, the information about a polynomial is stored in an ArrayList which emphasizes the OOP style and makes it easier to access a particular Term in the ArrayList and even its coefficient and degree.

3.3. Relationships

The classes which contain methods that execute the desirable operations are called in the GUI class, when the corresponding button is pressed. Moreover, GUI class contains two objects of type Polynomial, in which the information received from the user is stored.

3.4. Algorithms

The addition of two polynomials is performed by adding the coefficients of each two terms with the same degree. Firstly, the second polynomial is copied in a local newPol variable of type Polynomial. If the second polynomial has terms with a degree that does not have a correspondent in the first polynomial, those terms are simply added in the final result polynomial.

The subtraction is based on the same logic as the addition, with the only difference that the second polynomial is copied in the newPol variable with its coefficients negated.

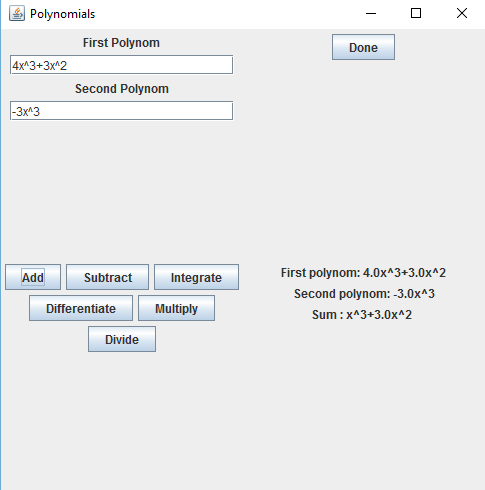
The product of two polynomials is done by multiplying the coefficients of any two terms (one in the first polynomial and the other in the second one). Because there are good chances that the result of this multiplication will contain terms with same degree, an addition is performed on this result.

The Euclidean division of two polynomials involves two additional variables of type Polynomial: quotient and remainder. This algorithm uses the addition, subtraction and multiplication presented before.

The differentiation of a monomial is done by multiplying the coefficient’s value with the degree’s value minus one and by subtracting one from the degree’s value. This operation is repeated for all the terms in a polynomial.

The integration of a monomial is done by dividing the coefficient’s value at the degree’s value plus one and by increasing the degree’s value. As for differentiation, this process is repeated for all monomials.

3.5. User Interface



As shown in the example, the user has to enter the first and the second polynomial in the corresponding text fields. In order to do that, the user has to insert the coefficient of a monomial, followed by the letter “x” or “X” and then by the specific sign “^” which suggests that the power of “x” is the next to be inserted. After pressing the “Done” button the two polynomials will be displayed on the screen, in a correct mathematical format.

The next step for the user is to choose the desired operation by pressing one of the buttons: “Add”, “Subtract”, “Integrate”, “Differentiate”, “Multiply”, “Divide”. The result of the chosen operation will be labeled in order to remind the user what was the last operation that was performed.

1. Implementation and testing

After pressing the “Done” button the polynomials entered by the user are split into monomials, which are added as distinct terms into the variables polynomeOne and polynomeTwo of type Polynomial, using the following algorithm:

Pattern p = Pattern.compile("(-?\\b\\d+)[xX]\\^(-?\\d+\\b)");

Matcher m1 = p.matcher(polynomText1.getText());

Matcher m2 = p.matcher(polynomText2.getText());

while (m1.find()) {

polynomeOne.addTerm(new Term(Integer.parseInt(m1.group(1)), Integer.parseInt(m1.group(2))));

}

while (m2.find()) {

polynomeTwo.addTerm(new Term(Integer.parseInt(m2.group(1)), Integer.parseInt(m2.group(2))));

}

The method addTerm(term term) is implemented in class Polynomial, as shown below:

public void addTerm(Term term) {

boolean add = false;

for (Term t : this.terms) {

if (term.getDegree() == t.getDegree()) {

add = true;

t.setCoeff(term.getCoeff() + t.getCoeff());

}

}

if (!add) {

terms.add(term);

}

}

One can observe that if the new term has the same degree as a term that is already in the ArrayList, the coefficient of the old term is added with the coefficient of the new term. Otherwise, the new term is introduced at the end of the list.

Each of the other buttons from GUI class corresponds to an operation. Below, there is presented what happens when the “Add” button, represented by “buttons[1]“, is pressed by the user:

if (source.equals(buttons[1])) {

AddOp add = new AddOp();

secondResult.setText(null);

firstResult.setText("Sum : " + add.execute(polynomeOne,polynomeTwo).toString());

}

One can see that an object of type AddOp is created in the actionListener of this button and then the execute(polynomeOne, polynomeTwo) method is called with the required parameters. The final result is printed using the toString() method, from class Polynomial.

There are two interfaces, both containing a method with the same name, but with different number of parameters:

public interface BiOperation {

public Polynomial execute(Polynomial p1, Polynomial p2);

}

public interface UniOperation {

public Polynomial execute(Polynomial p);

}

The BiOperation interface is implemented by the classes AddOp, SubtractOp, MultiplyOp, DivideOp. The implementation of the execute(Polynomial p1, Polynomial p2) from class AddOp is shown below:

public class AddOp implements BiOperation {

@Override

public Polynomial execute(Polynomial p1, Polynomial p2) {

Polynomial newPol = new Polynomial();

**for** (Term t : p2.getTerms()){

newPol.getTerms().add(**new** Term(t.getCoeff(),t.getDegree()));

}

for (Term term1 : p1.getTerms()){

newPol.addTerm(term1);

}

newPol.sortPolynom();

return newPol;

}

}

In order to subtract two polynomials, instead of copying p2 into newPol all terms from the second polynomial are copied into the new variable with their coefficients negated, as in the example:

for (Term t : p2.getTerms()) {

if (t.getCoeff() != 0) {

newPol.getTerms().add(new Term(-t.getCoeff(), t.getDegree()));

}

}

The multiplication is done by making the product between every term of the first polynomial with all the terms in the second one:

public class MultiplyOp implements BiOperation {

@Override

public Polynomial execute(Polynomial p1, Polynomial p2) {

Polynomial newPol = new Polynomial();

for (Term t1 : p1.getTerms()) {

for (Term t2 : p2.getTerms()) {

newPol.addTerm(new Term(t1.getCoeff() \* t2.getCoeff(), t1.getDegree() + t2.getDegree()));

}

}

newPol.sortPolynom();

return newPol;

}

As mentioned before, the division is made by using the Euclidean algorithm:

private Polynomial remainder = new Polynomial();

@Override

public Polynomial execute(Polynomial p1, Polynomial p2) {

// TODO Auto-generated method stub

Polynomial quotient = new Polynomial();

Polynomial aux = new Polynomial();

MultiplyOp multiply = new MultiplyOp();

SubtractOp subtract = new SubtractOp();

AddOp add = new AddOp();

int degree = p2.polynomDegree();

double lc = p2.leadingCoeff();

**for** (Term t : p2.getTerms()){

remainder.addTerm(t);

}

while (remainder.polynomDegree() >= degree) {

aux.getTerms().add(new Term(remainder.leadingCoeff() / lc, (remainder.polynomDegree() - degree)));

quotient = add.execute(aux, quotient);

remainder = subtract.execute(remainder, multiply.execute(aux, p2));

aux.getTerms().clear();

}

remainder.sortPolynom();

quotient.sortPolynom();

return quotient;

}

}

The classes that implement UniOperation interface are IntegrateOp and DeriveOp. Taking as an example the method from IntegrateOp class, we have the following code:

public class IntegrateOp implements UniOperation{

public Polynomial execute(Polynomial p) {

Polynomial newPol = new Polynomial();

for (Term term : p.getTerms()){

newPol.getTerms().add(new Term(term.getCoeff() / term.getDegree() + 1), term.getDegree() + 1));

}

newPol.sortPolynom();

return newPol;

}

}

The only difference in code for derivation is the following line:

newPol.getTerms().add(new Term(term.getCoeff() \* term.getDegree(), term.getDegree() - 1));

1. Results

We assume that the user introduces the following polynomials:

First polynomial: 1x^2 – 10x + 21x^0

Second polynomial: 1x^1 – 7x^0

On the screen will be printed:

First polynomial: x^2 – 10.0x + 21.0

Second polynomial: x – 7.0

The results for each of the operations will be:

* Addition: x^2 – 9.0 x + 14.0
* Subtraction: x^2 – 11.0 x + 28.0
* Multiplication: x^3 – 17.0 x^2 + 91.0 x^1 – 147.0
* Division: Quotient: x – 3.0

Remainder: 0

* Integration of the first polynomial: 0.33 x^3 – 5.0 x^2 + 21.0 x
* Integration of the second polynomial: 0.5 x^2 – 7.0 x
* Derivation of the first polynomial: 2.0 x – 10.0
* Derivation of the second polynomial: 1

1. Conclusion and feature developments

This problem was truly helpful in understanding the usefulness of the OOP style. However, it could use a few improvements, such as:

* Let the user introduce the polynomials in an easier way, without having to write degree and coefficient of value 1 or 0.
* Add more operations which can be done on polynomials
* Display the result of the integration with coefficients in shape of fractions, not real numbers
* Remove from the ArrayList of Terms those monomials with null coefficients, instead of not printing them in the final result

1. References

* <http://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial>
* <https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor#Euclidean_division>