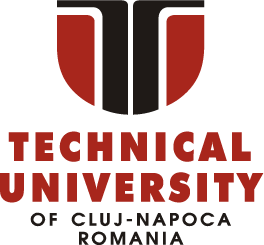
Technical University of Cluj-Napoca

Programming Techniques

Laboratory - **HOMEWORK 1**

Working with polynomials

**

*Teacher*: Ioan SALOMIE *Student*: Iulia Maria ṢOMFELEAN

*Teacher Assistant*: Delia BALAJ Group 30425

1. Objective

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

1. Dimensions of the problem

***A polynomial is a mathematical expression involving a sum of powers in one or more***[***variables***](http://mathworld.wolfram.com/Variable.html)***multiplied by***[***coefficients***](http://mathworld.wolfram.com/Coefficient.html)***.***

* 1. Analyzing the problem

Polynomial algorithms are at the core of classical "computer algebra".

A polynomial in one variable (i.e., a *univariate* polynomial) with constant coefficients is given by

displaymath60704

where an is different from 0.

The term ai is the coefficient of the ith power of *x*. We shall assume that the coefficients are integer numbers. The *degree* (also called order) in an univariate polynomial is given by it’s highest power. The individual summands with the coefficients included are called *monomials*.

An alternative representation for such a polynomial consists of a sequence of ordered pairs:

displaymath60705

Each ordered pair, (ai, i), corresponds to the term ai\*xi of the polynomial. An ordered pair is composed by the coefficient of the *ith term* and it’s index representing the power, *i*. For example, the polynomial *2+39\*X+54\*X2* can be represented by the sequence {(2, 0), (39, 1), (54, 2)}.

Consider now the 98th -order polynomial X98+5. Clearly, there are only two nonzero coefficients: a98  and a0 . The advantage of using the sequence of ordered pairs to represent such a polynomial is that we can omit from the sequence those pairs that have a coefficient equal to zero. We represent the polynomial X98+5 by the sequence {(1, 98), (5, 0)}.

So we can conclude saying that a polynomial is composed by one or more monomials ai \* xi , with *i* in the range [0, n], *n* being the degree of the polynomial.

This way of representing polynomials can be used to perform the most common operations on polynomials : addition, subtraction, multiplication, division, differentiation, integration and computing the value of a polynomial at a given point *x*.

* 1. Modelling the problem

The user will be able to introduce, through an user interface, two polynomials or a point *x*, and perform specific operations on them, such as:

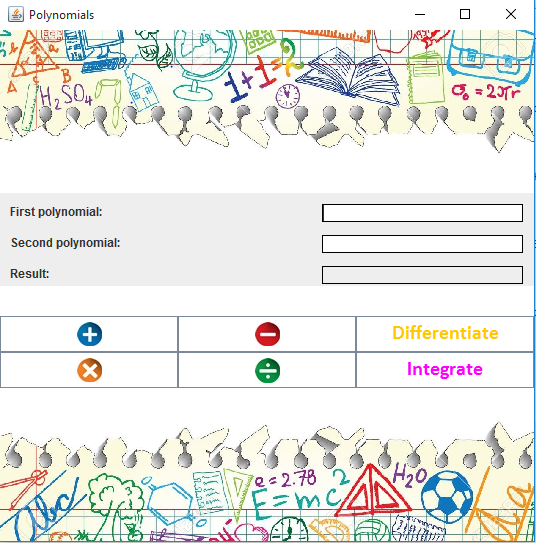
* Adding two polynomials
* Subtracting two polynomials
* Multiply two polynomials (this includes also the multiplication of a polynomial with a scalar, because an order zero polynomial represents a constant)
* Divide two polynomials
* Differentiate a polynomial
* Integrate a polynomial
* Evaluating the polynomial in a given point *x*

The obtained result will be displayed as a feedback to the user.

* 1. Scenarios and use cases

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

The use cases are strongly related to the user steps. I tried to design my interface in a user friendly mode, and that’s the result:



The user can introduce two polynomials, in a valid format, if they want to perform the +, -, \*, / operations, or only one polynomial in order to perform differentiation or integration.

The result will be displayed on the non-editable Result TextField.

The user must pay attention to follow the format of the polynomial.

1. Implementation
   1. Diagrams
2. Use case diagrams

Introduces data Performs operations on it

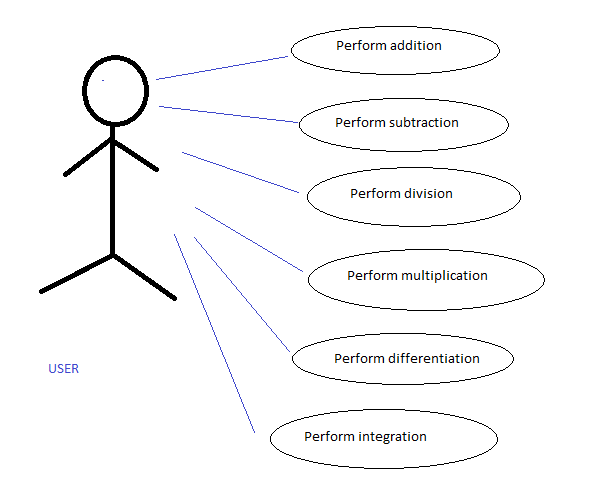
User

Developer

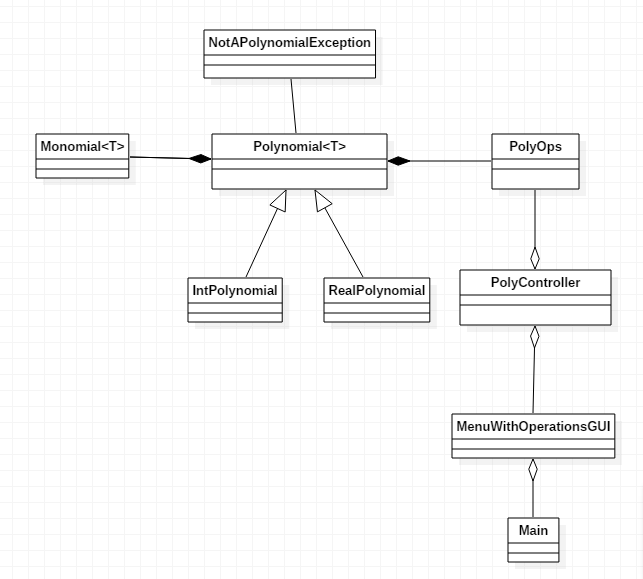
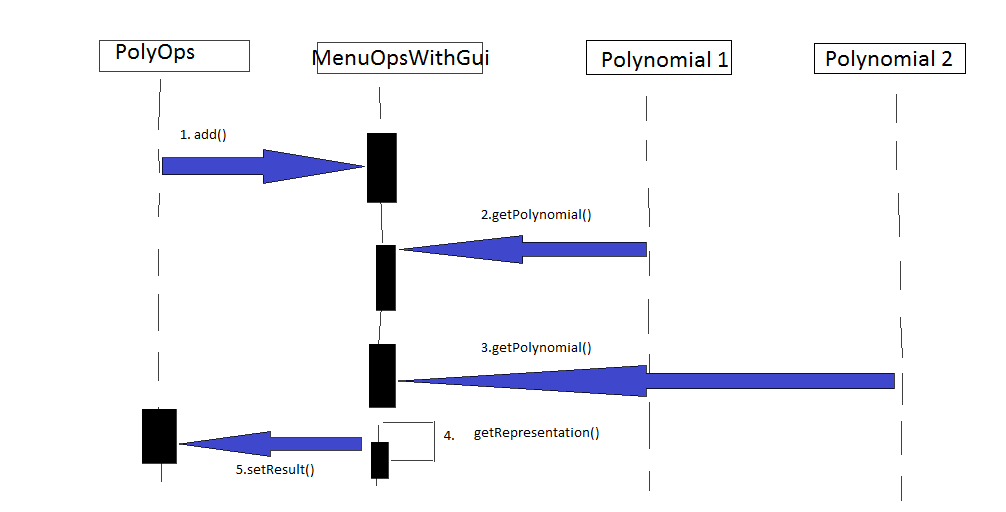
Maintenance

Improvements

Visualizes results



The use case diagram present the actor, which is the user that interacts with the application. He can perform several actions such as adding, subtracting, multiplying, dividing, integrating, differentiating polynomials.

1. Class diagram
2. Sequence diagram
   1. Data Structures

The data structures used in this problem are either primitive data types, mostly *integers* and *doubles* and a more complex one, such as *ArrayList* type objects or new created objects such as Monomial, Polynomial, etc.

Regarding this, I have decided to use ArrayList instead of classic arrays because I think that they are more efficient from the point of view of memory management, performance and provide a faster access to their content. Also, the size of an ArrayList is not fixed and they can be used with Generics.

Java **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively.

I chose to use Generics because the coefficients of a polynomial can integer or real numbers. According to that and to the fact that a polynomial is composed from monomials, I created a Monomial generic class, which is used by the Polynomial class. The object Monomial<T> was introduced for representing a term in the sequence of terms that form a polynomial. This terms were added in a list of type ArrayList<Monomials<T>>.

* 1. Packages

Java packages help in organizing multiple modules and group together related classes and interfaces. Packages avoid name conflicts.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

* A *Model* , which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.
* A *View* , which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)
* A *Controller* , which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

Given the fact that I have chosen a Model-View-Controller Pattern, I splitted my classes into 4 packages:

* **main:** contains a single class, Main, which contains the customary main() method.
* 
* **model**: contains the “brain” of the project, the classes that model the problem.
  + 
  + 
  + 
  + 
  + 
  + 
* **view**: represents the GUI
* 
* **controller**: the controller part interconnects the model and the view
* 
  1. Class Design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems .  
Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

I tried to design my project in the MVC architecture, that’s why I have 3 principal parts:

* + 1. The model – contains the logic of the application
       - **Monomial<T> Class**

A polynomial is composed by one or more terms, and we call that terms as monomials. A monomial has two fields: coefficient and exponent.

In our situation, T is a generic type corresponding to the coefficient which can take an integer value or a double value. The exponent is always an integer.

Constructor:

* *public Monomial(T coefficient, int exponent)* : the constructor that initializes the monomials with the transmitted coefficient and exponent.

Methods:

* *public T getCoefficient()* - Gets the coefficient of this monomial. Return coefficient the coeff of this monomial.
* *public int getExponent() -* Gets the exponent of this monomial. Return exponent the exp of this monomial.
* *public void setCoefficient(T coefficient)* - Modify the coefficient of the monomial. Parameter coeff the new coeff.
* *public void setExponent(int exponent) -* Modify the exponent of the monomial. Param exp the new exp.
  + **NotAPolynomialException Class**

This class extends the Exception class and consists in only one constructor that has one parameter: the message string.

The class is a user defined exception, thrown in the cases in which the polynomial received as an input it is not valid and operations cannot be performed accurately.

* **Polynomial<T> Class**

This class is abstract and has only one field, an instance of the List<Monomials<T>> class that is used to keep the monomials of this polynomial. The monomials are ordered in this list, from the lowest exponent of the polynomial to highest exponent of the polynomial.

Consider using abstract classes if any of these statements apply to your situation:

* You want to share code among several closely related classes.
* You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
* You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

Constructor:

* *public Polynomial()* - Create a polynomial object with no monomials

Abstract methods:

- *public abstract boolean addToMonomialsList(T coefficient, int exponent):* this method intends to add one more element to the monomials list

*- public abstract String printPoly() :* this method return a printable form of the polynomial

Methods:

*- public List<Monomial<T>> getMonomials() –* return the polynomial’s list of monomials.

*- public void setMonomials(List<Monomial<T>> monomials)* – sets the polynomial’s list of monomials.

**-** *public int getDegree()* - Returns the degree of the polynomial.

- *public Monomial<T> getMonAtExp(int exponent)* - Returns the monomial having the exponent given as parameter.

*- public ListIterator<Monomial<T>> iterator()* – Returns a list iterator for the monomials list

**IntPolynomial Class**

Extends the Polynomial<T> class, but instantiates the T with Integer type.

It is useful for the polynomials with integer coefficients.

Inherits all methods and instance variables from Polynomial<T>.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A subclass inherits all the *members* (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

Constructor:

* *public IntPolynomial() –* parameterless constructor, just invokes the constructor of the superclass
* *public IntPolynomial(String poly) throws NotAPolynomialException* - Creates a polynomial from a string received from input and throws an exception if the input is not valid

Methods:

* *public boolean addToMonomialsList(Integer coefficient, int exponent)* – adds to the list of monomials a new one with the given coefficient and exponent and returns true or false depending if the operation succeeded or not
* *public String printPoly() –* returns a printable form of the polynomial

**RealPolynomial Class**

Extends the Polynomial<T> class , but instantiates the T with Double type.

It is useful for the polynomials with real coefficients.

Inherits all methods and instance variables from Polynomial<T>.

Constructor:

* *public RealPolynomial() –* parameterless constructor, just invokes the constructor of the superclass
* *public RealPolynomial(String poly) throws NotAPolynomialException* - Creates a polynomial from a string received from input and throws an exception if the input is not valid

Methods:

* *public boolean addToMonomialsList(Double coefficient, int exponent)* – adds to the list of monomials a new one with the given coefficient and exponent and returns true or false depending if the operation succeeded or not
* *public String printPoly() –* returns a printable form of the polynomial
* *public boolean isEqual(RealPolynomial poly2) -* Checks whether a polynomial is equal or not with the polynomial received as parameter*.*
* *public boolean isEqualToZero()* - Checks whether a real polynomial is equal or not with zero.

**PolyOps Class**

* contains static methods for performing addition, subtraction,multiplication, division, integration and differentiation on real and integer polynomials.
* here are implemented the algorithms discussed
  + 1. The controller – contains the linking between the model and the view of the application.

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

**PolyController Class**

Has 2 fields : **private** MenuWithOperationsGUI view;

**private** PolyOps model;

Constructor:

* ***public*** *PolyController(PolyOps model, MenuWithOperationsGUI view)*

This class contains the action listeners for the buttons from the user interface: AdditionListener, SubtractionListener, MultiplicationListener, DivisionListener, DifferentiationListener, IntegrationListener.

* + 1. The view – View represents the visualization of the data that model contains.

Contains fileds that represent graphic elements, such as:

* ***private*** *JLabel background;*
* ***private*** *JButton addButton, subButton, mulButton, divButton, diffButton, integrButton;*

Constructor

* ***public*** *MenuWithOperationsGUI(String nameOfFrame)* – initializes the frame name, sets frame properties, like size or visibility, and add graphic elements on the container. As layouts I used mostly GridBagLayout() because it seemed to me the most customizable. I chose to put a background image to the interface, and I created a label with my desired background image, and I added up the rest of the elements on top of this label.

Methods

ActionListener added for all the buttons.

* ***public*** *String getFirstPolynomial()* : returns the first input polynomial as a String
* ***public*** *String getSecondPolynomial()* : returns the second input polynomial as a String
* ***public*** *void setResult(String res):* updates the content of the corresponding TextField whenever the result changes
  + 1. The Main Class
* contains the main method, which include a instantiation of the view and creates a controller.
  1. Algorithms

1. **Addition**

The sum of two polynomials is obtained by adding together the coefficients sharing the same powers of variables so, for example,

|  |  |
| --- | --- |
| (a_2x^2+a_1x+a_0)+(b_1x+b_0)=a_2x^2+(a_1+b_1)x+(a_0+b_0) |  |

and has order less than (in the case of cancellation of leading terms) or equal to the maximum order of the original two polynomials.

1. **Subtraction**

The sum of two polynomials is obtained by subtracting the coefficients sharing the same powers of variables so, for example,



and has order less than (in the case of cancellation of leading terms) or equal to the maximum order of the original two polynomials.

1. **Multiplication**

The product of two polynomials is obtained by multiplying term by term and combining the results, for example

|  |  |  |  |
| --- | --- | --- | --- |
| (a_2x^2+a_1x+a_0)(b_1x+b_0) | = | a_2x^2(b_1x+b_0)+a_1x(b_1x+b_0)+a_0(b_1x+b_0) |  |
| http://mathworld.wolfram.com/images/equations/Polynomial/Inline9.gif | = | a_2b_1x^3+(a_2b_0+a_1b_1)x^2+(a_1b_0+a_0b_1)x+a_0b_0, |  |

and has order equal to the sum of the orders of the two original polynomials.

1. **Division**

The division of the two polynomials is used using the long division polynomial algorithm, which has the following pseudocode:

1 function n / d:

2 require d ≠ 0

3 (q, r) ← (0, n) # At each step n = d × q + r

4 while r ≠ 0 AND degree(r) ≥ degree(d):

5 t ← lead(r)/lead(d) # Divide the leading terms

6 (q, r) ← (q + t, r - (t \* d))

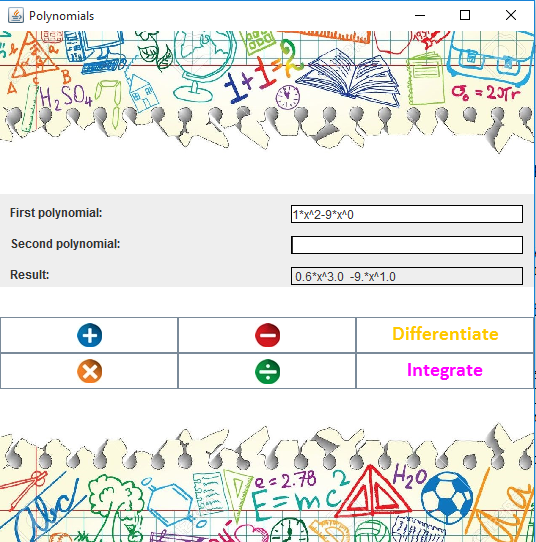
7 return (q, r)

1. **Differentiation**

A very simple algorithm to differentiate a polynomial which is represented by a sequence of ordered pairs was used:

1. Drop the ordered pair that has a zero exponent.
2. For every other ordered pair, multiply the coefficient by the exponent, and then subtract one from the exponent.
3. **Integration**
4. The algorithm checks whether the exponent is zero. If so, it only increments the exponent.
5. Otherwise, it multiplies the coefficient with the exponent and divides it with incremented by one exponent.
   1. User Interface

The user interface has the role of connecting the user with our application.



The user has the ability to introduce some input data, respectively one or two polynomials, and perform on them the desired operations available in the menu.

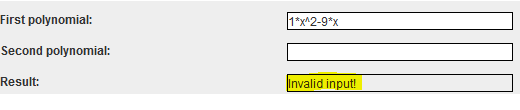
The input data has to fulfill some conditions to provide a correct functionality to our application, respectively:

* Every introduced polynomial is composed by one or more terms
* Each term must cover this format **[sign][coefficient][\*x^][exponent]**
* The square brackets must be ignored, they are used just for to make the format more visible and clear
* Even *coefficients equal to one* and *exponents equal to zero* must be specified in the input

1. Implementation and testing

This application was developed and tested only in Eclipse, but this thing should not affect it’s portability. From the point of view of the used algorithms, they are simple ones, taken from mathematics and implemented in Java.

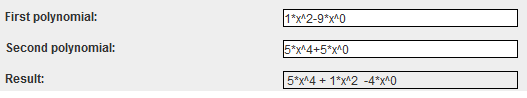
If the user enters an invalid input:



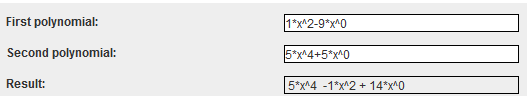
The correct way would be specifying also the zero coefficient, as it has to respect the format:



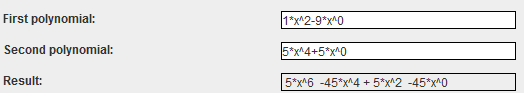
An example of addition:



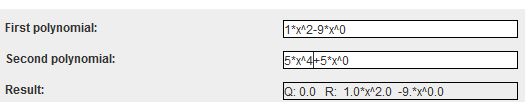
An example of subtraction:



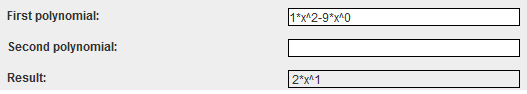
An example of multiplication:



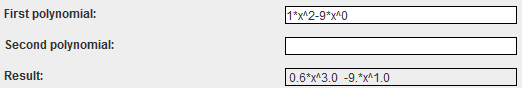
An example of division:



An example of differentiation:



An example of integration:



1. Results and improvements

The application is an user friendly and helpful application to perform basic polynomial operations, as long as the user obeys the input conventions and it is familiar with polynomial operations such as: addition, subtraction, multiplication, division, differentiation and integration. As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems (as long as they have the Java SDK installed).

Altough at the moment I have chosen to implement just the basic polynomial operations, this application can be improved by adding new functionalities, such as:

* Compute the square of the polynomial
* Compute the value of a polynomial in a certain point
* Find the roots of the polynomial
* Plot the graphic of the polynomial

1. Conclusions

This project was a good exercise in remembering and deepen the OOP concepts learned in the first semester, but also learning new ones. There are a few learned things which I would present next. First of all, time management is crucial, because a good organizatory spirit helps you see things gradually. Secondly, modeling the problem in a right way from the beginning helps you to implement it faster. Also, it is good to start with UML diagrams, to have an overall idea of the design, because it is much easier to implement it later. Thirdly, I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through th mean of research, has the benfit of learning new concepts and a better use of the known ones.

1. Bibliography
   1. Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
   2. Programming Techniques – Lectures of prof. Ioan SALOMIE
   3. Head First Java 2nd Edition, Kathy SIERRA
   4. [www.stackoverflow.com](http://www.stackoverflow.com)
   5. [www.math.stackexchange.com](http://www.math.stackexchange.com)
   6. [www.wikipedia.org](http://www.wikipedia.org)
   7. <https://reference.wolfram.com/>
   8. <http://whatis.techtarget.com/definition/model-view-controller-MVC>