*Technical University of Cluj-Napoca*

*Faculty of Automation and Computers*

*Department if Computer Science*

*2nd Semester 2015-2016*

*Programming Techniques*

*Homework 1*

Student: Hitter András

Group: 30424

Contents

[1. Problem Specification 3](#_Toc445772008)

[2. Example of working 4](#_Toc445772009)

[3. Design 5](#_Toc445772010)

[3.1. Relational Diagram 5](#_Toc445772011)

[3.2. Class Design 7](#_Toc445772012)

[3.3 Packages and Interfaces 13](#_Toc445772013)

[4. Using and testing the application 14](#_Toc445772014)

[5. Conclusions 17](#_Toc445772015)

[6. References 17](#_Toc445772016)

Abstract

In mathematics , a polynomial is an expression consisting of variables and coefficients that involves only the operations of addition , subtraction , multiplication and non - negative integer exponents . An example of a polynomial with of a single variable, x , is *x*2 − 4*x* + 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 functions . In advanced mathematics , polynomials are used to construct polynomial rings and algebraic varieties , central concepts in algebra and algebraic geometry .

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

a_n x^n + a_{n-1}x^{n-1} + \dotsb + a_2 x^2 + a_1 x + a_0,

where a_0, \ldots, a_n are numbers and X is a symbol which is called an indeterminate or variable . The symbol X does not represent any value , although the usual ( commutative , distributive ) laws valid for arithmetic operations also apply to it .

The same polynomial can be expressed more easily by using a summation notation :

\sum_{i=0}^n a_i x^i A polynomial can either be zero or can be written as the sum of a finite number of non - zero terms . Each term consists of a product of a number – called the coefficient of the term – and a finite number of indeterminates , raised to nonnegative integer powers. The exponent on an indeterminate in a term is called the degree of that indeterminate in that term ; the degree of term is the sum of the degrees of the indeterminates in that term , and the degree of a polynomial is the largest degree of any one term with a nonzero coefficient . The degree of an indeterminate without a written exponent is one . A term with no indeterminates are called respectively constant term and constant polynomial ; the degree of a constant term and of a nonzero constant polynomial is 0 .

# Problem Specification

Homework 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 some operations that can be done on one or two polynomials. Among these operations there are:

* addition, subtraction, multiplication and division of two polynomials
* derivation and integration of a polynomial
* polynomial representation using a string for display

The program will have and interface through which the user can communicate with the program. One can enter the data (in form: nx^n+n-1x^n-1….2x^1-1x^0 ) and after pressing the enter, or pressing the buttons (+, -, \*, /, integrate, derivate) the program evaluates the desired operation and also serves with an output in an understandable form.

The result will be displayed in a separate field highlighted with green color. In case of some operations (derivate, integrate) the program does the operation for both of the polynomials, and outputs the results in the first and second result fields according to the first and second entered polynomial.

# Example of working

Assumed that one entered the following polynomials (without “ ” ):

"3x^3+3x^2" (*press Enter*)

"-3x^2+2x^1" (*press Enter*)

(-*click on one of the buttons-*)

The results for the operations will be the following:

* Addition result: 3x^3+2x
* Subtraction result: 3x^3+6x^2-2x
* Multiplication result: -9x^5-3x^4+6x^3
* Division result: -x-1.66
* Derivation result for the first polynomial: 9x^2+6x
* Integration result for the first polynomial: 0.75x^4+x^3
* Derivation result for the second polynomial: -6x+2
* Integration result for the second polynomial: -x^3+x^2

The program will have an interface so the user can relatively easily introduce the desired inputs which are the two polynomials. The program gives feedback if the input is correct or not and how many inputs are needed. The user can click on buttons on which are present the symbols for addition, division, subtraction, multiplication, derivation, and integration. The result(s) will be displayed in the last two text fields, before the operation buttons after the statements “Results:” and “Result only for 2nd polynomial derive and integrate:”.

# Design

## Relational Diagram

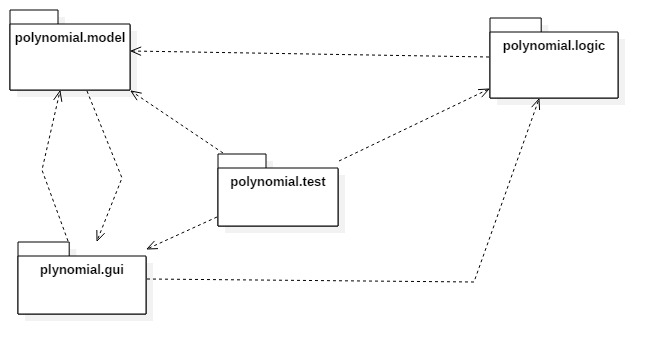
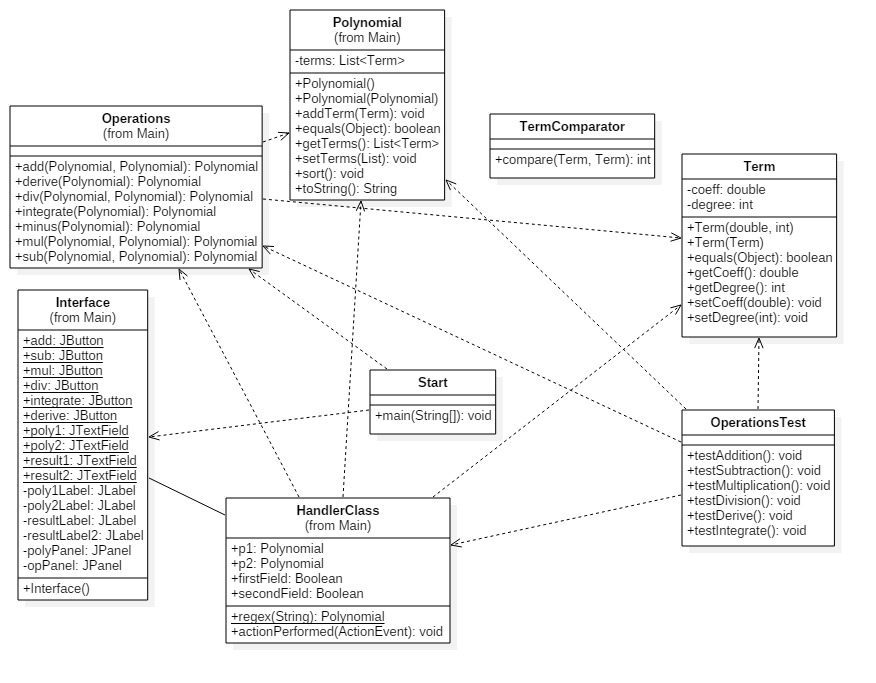
To resolve the problem, I’ve chosen to use four different packages, so that the design of application would be easier. Every package contains the classes which has strict semantical relation with it. For example the package: *polynomial.model* has inside it the classes *TermComparator, Term, Start* and *Polynomial* classes. Their names and dependencies are presented below:

Figure 1 Package diagram

From the picture above you can see the relationships between pakcages: all packages have relations of "dependency" between each other, which means that one’s class uses another’s or uses an object or a method of another’s class. The packeges has the following classes each:

* Polynomial.model: Polynomial, Term, TermComparator, Start(main())
* Polynomial.logic: Operations
* Polynomial.gui: HandlerClass, Interface
* Polynomial.test: OperationsTest

To get a better view related to the attributes of each class, there are below the UML diagrams for each class. Thus, we can see every class with objects and their methods.

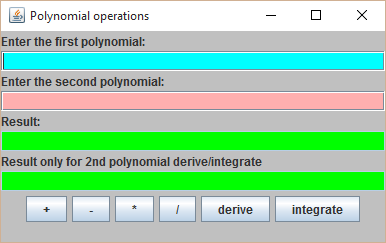


## Class Design

1. *Interface* Class : public class Interface extends JFrame

This class is designed to create a graphical user interface so the application would be easier to use. We consider the Interface class as a subclass of the predefined class "JFrame" so we can use objects of type "button", "frame" or "panel".

In this class there is only a constructor, by wich is realisied the creation of a container. To this container there are added more panels: polyPanel, opPanel. The previous one is responsible for containing -4- JTextFields in which the user may enter the polynomials -first two- and get the result(s) or further instructions -last two with labels “Result”- . The last one is responsible for containing buttons, for activating different methods of the program: add, sub, mul, div, integrate, derive. The program will not accept the command for the addition, subtraction, multiplication, division commands/buttons until the user doesn’ enter correctly 2 polynomials.

In case of derivation, integration it will do the same until at least 1 polynomial was not entered correctly. It works for one or two polynomials at the same time.

The attributes if the Interface Class:

To achieve the desired Interface I needed many attributes of different types:

* A JFrame: it is created by extending the JFrame
* Panels:
  + Private panel polyPanel – contains all the text boxes necessary to maintain the communication with the user
  + Private opPanel - contains all the buttons necessary which by the user interacting with the program can understand what he or she wants and execute the corresponding operation(s).
* 6 Buttons for executing the desired operations by the user:
  + Public static JButton add;
  + Public static JButton sub;
  + Public static JButton derive;
* Textfields:
  + Public static JTextField poly1;
  + Public static JTextField poly2;
  + Public static JTextField result1;
  + Public static JTextField result2;
* Constructor:
  + Public Interface() {…} – the initializations, coloring, placing(layout) are done here.

The following elements will be created: frame, textfields, buttons, labels.

* Creating a new HandlerClass object:
  + HandlerClass handler = new HandlerClass();

Every button is added to and ActionListener: add.addActionListener(handler);

div.addActionListener(handler); etc.

1. *HandlerClass* class: public class HandlerClass implements ActionListener;

This class was created to keep track of the user activity, and do operations corresponding to the user will. This class implements ActionListener class: public class HandlerClass implements ActionListener; It was required that I import the following classes: Operations, Polynomial, Term, in order to be able execute the operations according to the user input.

This class contains:

* Polynomial objects( attributes ):
  + Public Polynomial p1, p2 = new Polynomial();

The user input after using the regex(String a) method will be stored in these two objects, so I the program can work with them.

* ( attributes ) firstField, secondField Booleans which will let the program know if the polynomials were introduced correctly or not.
* The method regex(String):Polynomial

This method is capable of transforming a (string) user input into Polynomial (object). It helps the program understand that what the user wants to work with.

* The actionPerformed(ActionEvent): void method.

This method always listens to the buttons, textfields. If the user enters something this method reacts and performs operations predefined for the case that a particular button (or enter in a textfield) is pressed.

Also checks the correctness of the input. It will give instructions if an incorrect input occurs.

1. *Operations* class

This class is the heart of everything in this project! The operations class contains the implementation of all the operations with the polynomials.

In this class there is:

* NO CONSTRUCTOR
* 6 well implemented operations: add, sub, mul, div, derive, integrate
  + Public static Polynomial add(Polynomial p1, Polynomial p2)

This method gets two Polynomial type arguments. The first one is copied to a new object, because I wanted to use the same input for many operations, and without this copy the original input would be changed.

The second argument is iterated through and each one of it’s terms is added to the copy of the first argument. The returned result contains the addition of the two polynomials.

* + Public static Polynomial sub(Polynomial p1, Polynomial p2)

This method gets two Polynomial type arguments. The first one is copied to a new object, because I wanted to use the same input for many operations, and without this copy the original input would be changed.

The second argument is iterated through and each one of it’s terms is negated and after this, simply using the previous add(p1,p2) method are added to the copy of the first argument. The returned result contains the subtraction of the second polynomial from the first one.

* + Public static Polynomial mul(Polynomial p1, Polynomial p2)

This method gets two Polynomial type arguments. Both of them are iterated through in a double for loop and every Term from the first polynomial is multiplied with every term from the second polynomial. These terms are added to the result polynomial.

It is also necessary to sort the result polynomial, just to be sure that it is in the correct form.

* + Public static Polynomial div(Polynomial p1, Polynomial p2)

This method gets two Polynomial type arguments. It is tested that the second polynomial is not equal with zero. If this condition is met, the method starts working.

While the degree of the first polynomial is larger then the second one’s and the remainder is greater than zero, the method searches for the term with the largest degree in both of the polynomials.Remainder = p1. After it is found the quotient is calculated, and stored in a temporary result and the finalResult. Result \* second polynomial = remainder. So on and so forth the result is obtained.

* + Public static Polynomial derive(Polynomial p1)

This method receives only one argument, and calculates term by term its derivative. The result is saved in the result variable.

* + Public static Polynomial integrate(Polynomial p1)

This method receives only one argument, and calculates term by term its integrate. The result is saved in the result variable.

* + Public static Polynomial minus(Polynomial p1)

It gives to all the p1’s terms a negative sign.

1. *Polynomial* class

This class is the modelling of a real polynomial. It has: a data structure called terms ( private List<Term> terms ) :

* Terms:

This data kind of data type is formed in the Term class. The polynomial class stores a list of many term elements, and in this way creates an object very similar to a polynomial.

* It has two constructors:
  + Public polynomial(){…} which is an empty polynomial object. It creates the arraylist and is ready for using that.
  + Public polynomial( Polynomial ){…} this one is very useful when I want to copy a polynomial into another.
* Public void Sort()

This method sorts a polynomial ( Decreasing after degree )

* Public void addTerm(Term t)

This method will add a new term into an existing polynomial. Checks if a term with that degree already exists in that polynomial. If no creates a new term with this property, else adds together the coefficients.

It is the heart of the Operation.add( p1 , p2 ) method. If a new term + an old one equals 0, then that term will be removed from the equation, because we do not care about 0 elements.

* Public Boolean equals(Object obj)

This method is capable for comparing 2 different objects. For example two polynomials, or terms with it’s help is possible to check if we divide with 0 polynomial or not.

* Public List<Term> getTerms()

This method is a getter. If one asks for a polynomial terms it will be useful.

* Public void setTerms(List<Term>)

This method is a setter. If one wants to set a plynomial’s term it will be helpful.

* Public String toString()

This method is essential to the program to be able to communicate with the user. This method translates a polynomial’s content and provides understandable String output.

1. *Start* class

The start class is a simple class. It contains the main method, in which the interface is created and called, so the application can begin it’s working time!

Interface go = new Interface() ;

1. *Term*

This class’s job is to modell a single term from the polynomial. A term has coefficient and degree.

* Private double coeff;
* Private int degree;

As almost every class, this one has constructors too

* Public term(double coeff, int degree)

We can create a new term with this: Term t = new Term( 3 , 2 );

* Public term(Term t)

One can copy a term on another like this: Term s = new Term(p);

* Get/Set methods – the set or return a term’s coefficient and degree

1. *TermComparator*

This class is very simple. Compares two terms and returns 1 if the first elements’s degree is smaller and returns -1 if the second element’s degree is smaller.

1. *OperationsTest*

This class has a JTestUnit for every operation. This means, that I provide an input, and the corresponding result for every operation. The testunit runs the method of and operation, and comparest it’s result with my result. If these two equals then it shows a green bar, which means everything is fine.

There is a separate Testunit for every operation. Example at the end of the documentation.

## Packages and Interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. A package provides a unique namespace for the types it contains. Classes in the same package can access each other's package-access members.

A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks.

For this application the following packages are imported, each of them having a certain role for the proper working of the application. We import them in the Gui Class (most of them relate to the user interface properties):

* import java.awt: Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.
  + java.awt.BorderLayout: A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.
  + java.awt.Color: The Color class is used encapsulate colors in the default RGB color space or colors in arbitrary color spaces identified by a [ColorSpace](http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/color/ColorSpace.html).
  + java.awt.Dimension: This encapsulates the width and height of a component (in integer precision) in a single object.
  + java.awt.GridLayout: The GridLayout class is a layout manager that lays out a container's components in a rectangular grid for a better view of all the buttons and textfields which are added to the main panel.
* import java.awt.event
  + java.awt.event.ActionEvent;
  + java.awt.event.ActionListener;
* import javax.swing: Typical Swing applications do processing in response to an event generated from a user gesture. For example, clicking on a JButton notifies all ActionListeners added to the JButton. That’s why we use this package for creating the user interface Gui.
  + javax.swing.JButton;
  + javax.swing.JFrame; javax.swing.JLabel; javax.swing.JPanel;

## Using and testing the application

In order to use the application open Homework1. This will open a window which generates the Gui class. Thus the user can enter the desired values and selecting the operations by pressing one of the 6 buttons.

Rules for the input data:

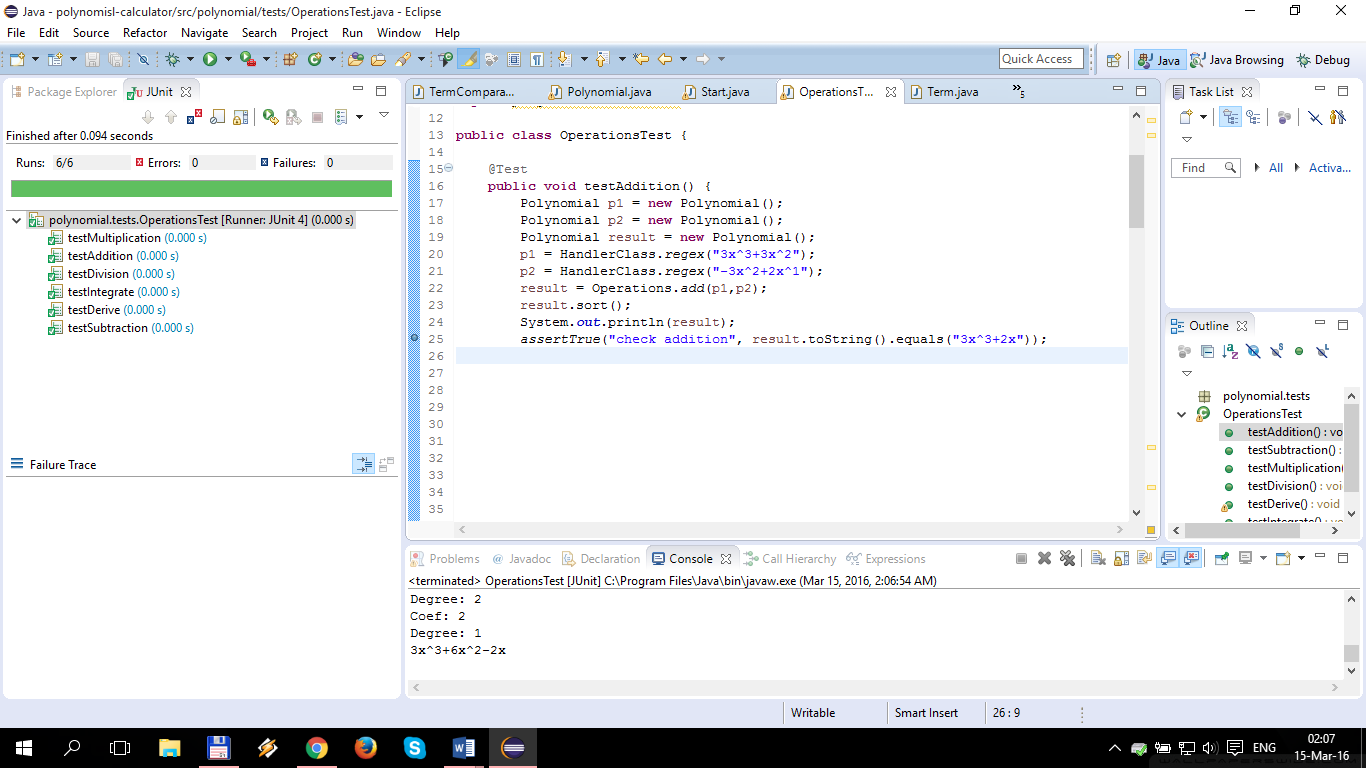
* The two polynomials must be given using the model of the following monomial:

sign + coefficient + ”x” + ”^” + exponential

Note: The sign must be mentioned even if it is ”+” because of the parsing condition. What is more, the coefficient for x^0 must be mentioned event if is null.

* Before executing the addition, subtraction, multiplication or division operations there must be a polynomial in the second field; otherwise this will be considered NULL.
* Do not input values too great to be tested; even if the application is design to work with polynomials of degree 100, the text field is not big enough to support a great amount of characters for displaying the result (at most 30 characters).

Testing examples:



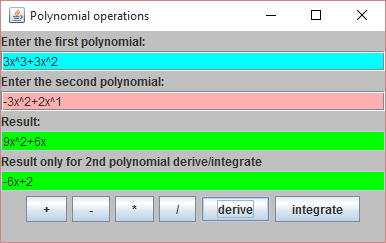
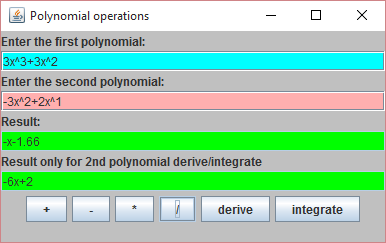
"3x^3+3x^2"

"-3x^2+2x^1"

Results:

* Addition: 3x^3+2x
* Subtraction: 3x^3+6x^2-2x
* Multiplication: -9x^5-3x^4+6x^3
* Division: -x-1.66
* Derivative: 9x^2+6x
* Integrate: 0.75x^4+x^3

The last two are done for the first polynomial.

Derivation and Division

# Conclusions

Achieving such a program may be hard both in terms of algorithms, graphical structure. Although some instructions were easy for implementation, like the addition or subtraction, division or integration operations were difficult and some problems in implementation occurred.

Moreover, to read and display a legible polynomial took a special function to transform the data type which was represented by a polynomial into a string of characters to be displayed or read.

For a better performance there should be implemented all cases where exceptions can occur and the application stops working due to an error made ​​by the user. It could handle more operations, for example ( polynomial )^n or it could work with more polynomials. Another thing that could be improved is the display so that it would be more elegant.

It was a really good practice, which taught me many things about this kind of programming.

# References

[*http://stackoverflow.com/*](http://stackoverflow.com/)

[*http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html*](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html)

*Teacher(s)*