Polynomials

Pop Roxana

Group 30424

**Contents:**

1. Objective………………………………………………………………………………………………………………………………….3

2. Problem Analysis………………………………………………………………………………………………………………………3

2.1 Data Modeling………………………………………………………………………………………………………………….3

2.2 Use Cases…………………………………………………………………………………………………………………………3

3. Design………………………………………………………………………………………………………………………………………4

3.1. Packages and classes………………………………………………………………………………………………………4

3.1.1. Model……………………………………………………………………………………………………………………….4

3.1.2. View………………………………………………………………………………………………………………………….8

3.1.3. Controller…………………………………………………………………………………………………………………..9

4. Implementation and Testing…………………………………………………………………………………………………….12

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

6. Conclusion…………………………………………………………………………………………………………………………………14

7. References………………………………………………………………………………………………………………………………..15

1. **Objective**

The purpose of this project is to be an implementation of a system for polynomial processing with integer coefficients.

1. **Problem Analysis**

The system could support at a simple level mathematical operations, both binary (addition, subtraction, multiplication and division), and unary (differentiation and integration) on polynomials of one variable.

The polynomials on which the operations are to be performed will have integer coefficients, as specified, but the result could have (depending on the operation) real coefficients.

* 1. Data Modeling

The problem domain revolves around the concept of the polynomial. Therefore, first one must abstract the polynomial, transform it in an object. A polynomial in one variable, say *x*, can be seen as a collection of terms, where each term is a pair coefficient - power of variable. That is, for a polynomial of the form

++…, a term is in the form , therefore defined through coefficient, , and degree, i.

* 1. 2.2 Use Cases

**Name**: Do polynomial operation

Description: Do an arithmetic operation on two or two polynomials

**Actor**: User

**Post condition**: The resulted polynomial will be displayed

**Basic Course of Action:**

1. The user inputs two polynomials and choses an operation

2. The operation is performed successfully and the resulted polynomial is displayed

**Alternate Course of Action 1:**

1. The user inputs the polynomials in the wrong format
2. The system displays an error and the user has to input the polynomials again

**Alternate Course of Action 2:**

1. The user inputs the zero polynomial as the second polynomial.
2. The user choses as the operation division
3. The system displays an error
4. The user has to input the polynomials again
5. **Design**
   1. Packages and classes

I approached the problem in an MVC manner. I divided the classes in mainly 4 packages: model, controller, view and utilities. The purpose of this approach is to separate the data and the interface, so that either of them can be modified with no changes being necessary in the other part.

* + 1. **Model**

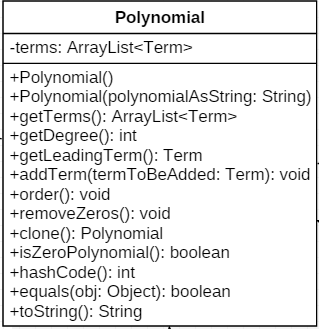
The model package contains the classes that are describing solely the model, i.e. the polynomial and the operations performed on it, and are therefore independent from those related to the visual interface.

The model package contains the classes:

* Polynomial;
* Term ;
* DataRetriever;
* a subpackage comparators :
* MyComparator;
* a subpackage exceptions :
* DivideByZeroException;
* InputFormatException;
* a subpackage operations:
  + PolynomialOperations;
  + TermOperations.

I will now take each class and explain its purpose and implementation.

3.1.1.1 Polynomial

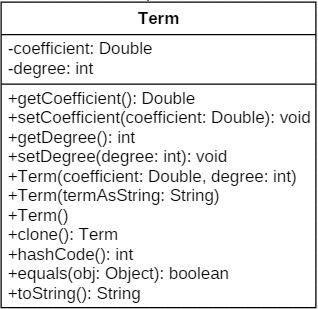
The polynomial class represents the main abstraction of the data. As it can be seen in the UML class diagram from bellow, a polynomial object consists from a collection of other objects, i.e. terms, and a set of methods.

The class has two constructors, a non-parameter one, and a constructor that transforms a String into a polynomial of type object. The second one is useful when linking the model to an interface, from where the data usually comes in the form of Strings.

It has methods that modify the object, as adding a new term, or ordering the polynomial, but also methods that retrieve information, like whether it is the zero polynomial, or what degree it has.

Even though operations like getLeadingTerm and isZeroPolynomial are not necessary for defining the concept of a polynomial there are rather useful in this particular implementation and still describe a polynomial.

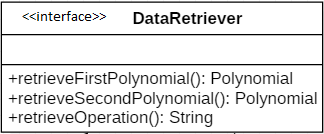
3.1.1.2 Term

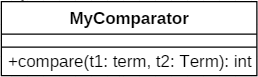
A term is the counterpart in the solution domain of a real world term object,. Therefore a term has as attributes the coefficient () and the degree of x (*i*).

The methods support a behavior like modifying the term (coefficient or degree) or transform it into a printable format (String). This class too has a constructor that receives a String, based on the same reasoning.

I chose to use Double for the coefficient as, even though the problem specifies to work on integer coefficients, the division of two polynomials, or the integration of one polynomial, are both operations that may result in real values as coefficients, and I wanted the system to support that. Nonetheless, the polynomials appear to have to the user, when possible, integer coefficients.

3.1.1.3 DataRetriever

This is an interface that allows the connection of the model to the interface. It simply has methods for retrieving the polynomials and the operation to be performed on them. It will be implemented by whichever class makes the link to the actual interface.

3.1.1.4 MyComparator

This class extends the Comparator interface. It allows ordering the collection of terms contained by Polynomial. The terms are ordered decreasingly by their degrees, meaning in the ordered polynomial is the first term.

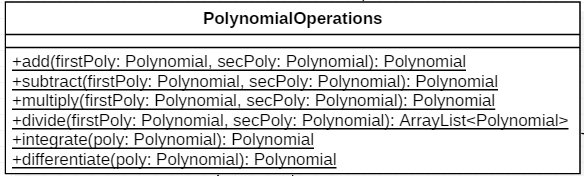
3.1.1.5 DivideByZeroException

This class extends the pre-defined Exception, and is therefore a user defined exception. It can be thrown by the divide operation, in case the divider is zero.

3.1.1.6 InputFormatException

This is another user defined exception, meant to be thrown in case a String cannot be converted into a polynomial.

3.1.1.7 Polynomial Operations

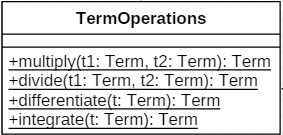


This is the class that manages the operations that are performed on the polynomials. It has static methods for addition, subtraction, multiplication, division, integration and differentiation. I chose to put all the methods in one class as it seemed logical to access them from the same place, something like accessing methods from the predefined *Math*. As visible from the class UML, all the methods are static so that there is no need to create an object of type PolynomialOperations in order to perform an operation on polynomials.

As for their implementation, all the methods besides divide were quite straight-forward. For addition I simply added copies of all the terms in a new polynomial, and similarly for the subtraction, with the remark that for the terms of the polynomial to be subtracted I reversed the sign. The multiplication consisted of adding into a new polynomial terms obtained by multiplying each term of the first polynomial to each term of the second polynomial. The integration and differentiation were obtained by integrating/differentiating the polynomials term-wise and adding the newly-created terms into new polynomials.

Now as for division, I tried to describe in an OOP style the usual algorithm: compute at each step a new term for the result by dividing the terms of the dividend and the divider, and also compute a new dividend by subtracting a polynomial from it.

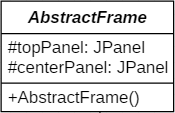
3.1.1.8 Term Operations

This class was helpful in further abstracting the methods in the PolynomialOperations class. Now many of the methods needed to perform the operation term-wise, and it only seemed natural to develop a class for that. Again, all methods are static, for easier access.

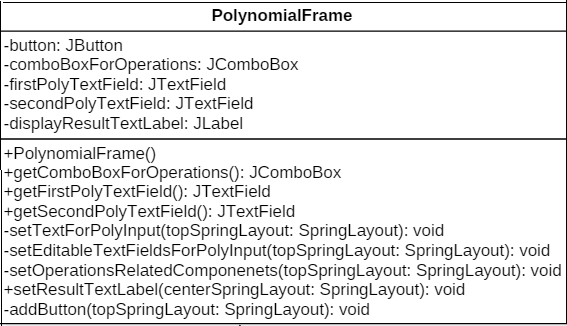
* + 1. **View**

The view package contains the classes that are describing the visual interface, more precisely classes that implement a visual interface using Java Swing. In addition to those there is a class that implements the DataRetriever interface from the model package, connecting the two. The classes are:

* AbstractFrame;
* PolynomialFrame;
* FrameDataRetriever;
  + - 1. AbstractFrame

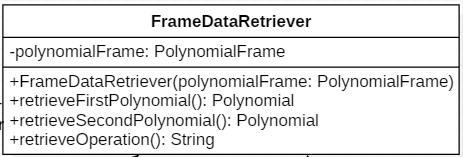
I chose to have an AbstractFrame that gives the general appearance of a window, in case the application uses more windows, and I didn’t want it to be project specific. Therefore, this class is not at all linked to the concepts of polynomials, it doesn’t provide fields for them or buttons with operations. It simply provides a two panel frame with some characteristics (size, close button, color) set.

* + - 1. PolynomialFrame



The PolynomialFrame is the class that gives the specific appearance and behavior of the interface and it actually extends the AbstractFrame. It has JTextLabels for displaying messages, but also the polynomial resulted, a JComboText for choosing the operation, a GO button for performing the operation and JTextFields for inputting the two polynomials. In order to avoid having a huge constructor in which all the elements would have been put on the frame, I chose to have some private methods to do that. I used such private methods for setting all the elements: text labels, text fields plus the button and combo box. To these methods I only sent the layout, as I used two different spring layouts, one for each panel of the abstract class. I also have public methods for retrieving certain elements as the controller needs the two polynomial text fields for retrieving the polynomials, the combo box for retrieving the operation and the result text label for outputting the resulted polynomial.

* + - 1. FrameDataRetriever

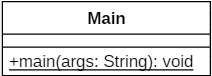


The FrameDataRetriever is a class that implements the DataRetriever interface present in the model part. I chose to place it in the view package as it is specific to this interface. As it can be seen from the UML diagram of the class from above, this class has as an attribute an object of type PolynomialFrame, that is actually passed through the constructor. The reason for that is the fact that in order to implement the methods, this class needs access to some items from the frame: the two text fields for retrieving the polynomials and the combo box for retrieving the operation. Moreover, this object couldn’t have been passed through the methods directly as it wouldn’t have corresponded to the methods signature in the interface implemented. From what I detailed here about the implementation, one could understand why I considered this class strongly linked to the interface and chose to add it in the view part.

As a further development, a class for polynomial frame data output that would extend a data output interface could be developed, thus adding better independence of the view from the rest of the project.

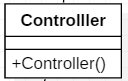
* + 1. **Controller**

The controller is the part that puts everything together. It consists of three classes: Main, Controller and ButtonActionListener. The mechanism is as follows: when a Java app is run, first runs the main method that is found in Main. There, an instance of the Controller has to be made, so that the control is actually passed to that object. The controller creates the instances needed to manage the whole project and keeps them in order, calling their methods and passing them around. Is usually the class that links not only other classes but parts of the project as well (view, model). Now in the controller section I found fit to place the classes whose code is triggered by events. As this is an interactive project, a lot of the controlling process is handled by action listeners, actually by the action listener, as there is only one. If there were more, they would all be placed in this section.

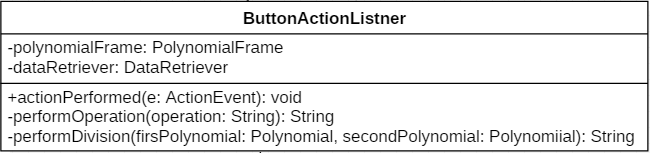
* + - 1. Main

This class contains only the main method where a new Controller is created.

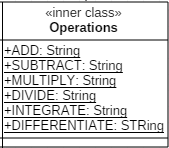
3.1.3.2 Controller

This class actually creates instances of the PolynomialFrame, FrameDataRetriever and ButtonActionListener. It creates the first instance to send it to the FrameDataRetriever, and then the second to send it to the ButtonActionListener. It doesn’t have any of them as fields as it doesn’t need any methods, the rest of the logic is left to the button listener as its code is executed when certain events trigger so.

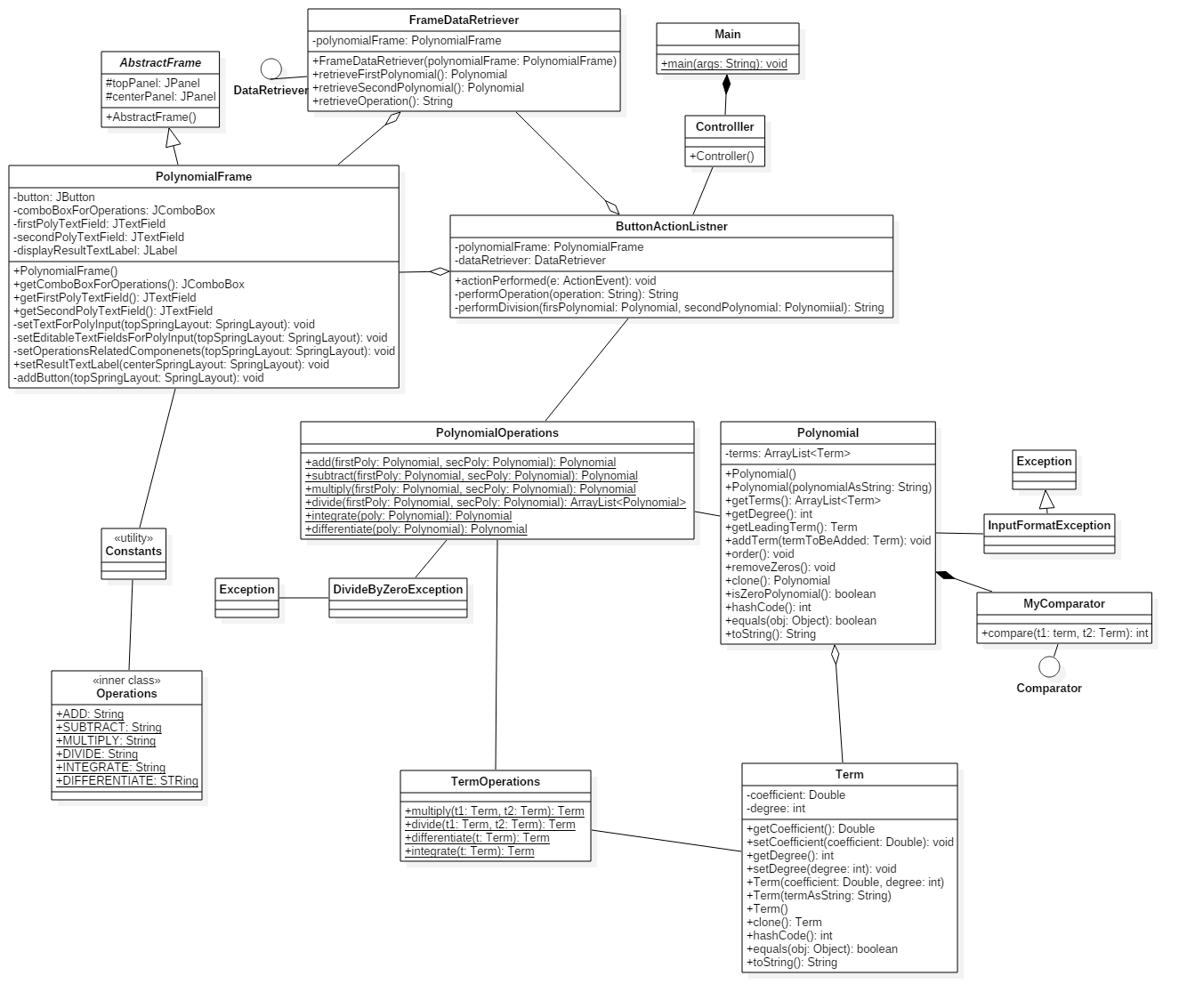
* + - 1. ButtonActionListener



This is the class that deals with the responsiveness of the program. At each click on the GO button, it retrieves the polynomials and the operation, with the help of the dataRetriever, performs the selected operation and outputs either the resulted polynomial, either an error message if something went wrong. To divide a bit the logic, it uses two private methods, one that returns the result as a string being given the operation, and one for performing solely the division.

I also have a package utilities where utilities classes are to be placed. As for now, the package consists of one class only, the Constants class. This has an inner class, Operations, that has as fields constants representing, oh well, the operations. By doing this I assured the name of the operations is stored in one place, and can be accessed from anywhere making the code both less error prone and more readable. The diagram of the inner class can be seen bellow:

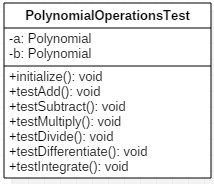
And this is how they are all linked together:



At the click of the GO button, the button listener retrieves the necessary data in order to perform the operation (the polynomials and the operation). When retrieving the polynomials, it deals with the wrong input format exception, and an error message is displayed if neccesssary .It then switches on the obtained operation and in each case calls a method from the PolynomialOperations. It calls it statically, so there is no need for extra instances. The division operation could also throw an exception, and this is dealt with as well, again, through an error message that is displayed to the user acknowledging him about the problem. Within this process, polynomial instances are passed a lot, and the conversion from String to polynomial assured by the constructor plus the toString method of the polynomial make the connection between the model and the interface easy.

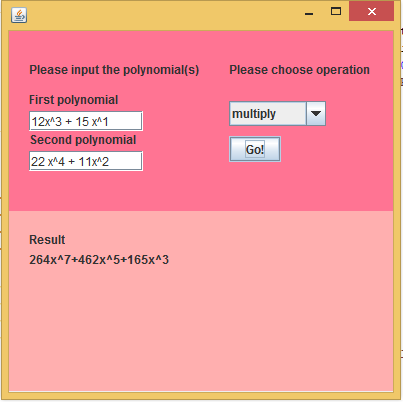
1. **Implementation and Testing**

The implementation part was already covered. As for the testing, I designed a class that would test each of the polynomial operations, using junit. The class has the following UML diagram:

As it can be seen in the left, the class has as attributes two polynomials: a and b. These two polynomials are initialized in the initialize method and then further used in testing almost all of the methods. The division is also tested on two other polynomials, actually two constants. I had to treat that case in the algorithm in particular so I needed to add that case in the testing part too, to make sure it worked properly.

1. **Results**

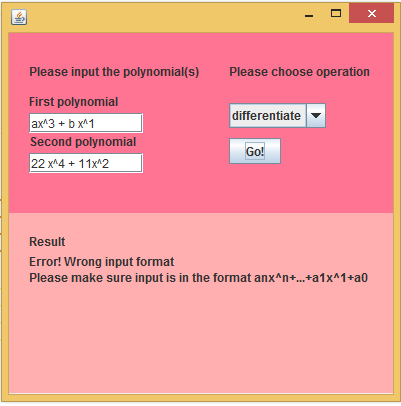
Leaving all the implementation details aside, this is what the application looks like for the user:



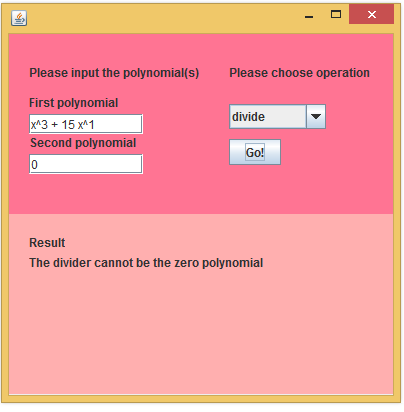
The polynomials are written in as plain text, an operation is selected and when the GO button is pressed, the resulted polynomial is written in the output (lower) section of the window. If a binary operation (that is performed on two operands is selected), the behavior is quite straight-forward. Now, when an unary operation is selected, that is, an operation to be performed on only one polynomial, the chosen polynomial is the first one, and the text field of the second one doesn’t disappear. Now this could be considered as a drawback of the current interface and, as a further development, it should be given a solution.

The interface is user friendly even in the case the user inputs a wrong polynomial format, or a zero divider when choosing the division. As specified in the use case, an error message is displayed corresponding to the particular problem and the user can input again the polynomials.

So, for wrong format input:



And for a zero divider:



1. **Conclusion**

I tried solving the given problem by a simple program that works on polynomials and provides also a basic user interface. The program performs as for now only basic arithmetic operations on polynomials (addition, subtraction, multiplication, division, differentiation and integration). In my attempt of solving the given problem, I tried to have as much as possible an OOP approach. That is why I viewed the polynomial as a collection of terms, rather than storing it as an array of coefficients, where each index would have been the degree of the polynomial variable. I also tried using abstract classes, interfaces and exceptions. Moreover, I attempted to work on the mvc model, thus assuring that the project is easily expandable.

As for what I learned from this assignment, working with user defined exceptions and unit tests would come off the top off my head. But leaving Java specifics aside, I think this homework was rather useful in exercising the modeling of a particular problem.

As for further developments, one could first of all improve the OOP design. The components could yet be given more independence, and that could be through the usage of more interfaces. For example, as there is an interface DataRetriever, there could be an interface DataDisplayer, that has a method displayResult which has no link to an actual implementation of an interface. And speaking of interfaces, the visual interface it as for now quite basic and not so much user friendly. I already mentioned the drawback of having the same appearance for the unary operations. And apart from that, the appearance could be improved and more options added. And if one were to add more options to the interface, those would have to be supported by the model implementation. As it is, the project is limited to the basic mathematical operations, but it could also provide features like finding the roots of the polynomial, and not only to first and second order polynomials, but of any order, through different approximation methods. Moreover, one could add drawing the graphic of the polynomial as a feature.

And the list of potential features could go on. The important fact is nonetheless that the project as it is could be considered as a solid base for such improvements.

1. **References**
2. <https://docs.oracle.com/en/java/>
3. <http://stackoverflow.com/>
4. <http://www.tutorialspoint.com/>
5. <http://agilemodeling.com/artifacts/systemUseCase.htm>