Documentation

Operations on Polynomials with integer coefficients

Student: Rusu Daniel
Group: 30422

Table of Contents

1. Objective

Main objective of the assignment:
Design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

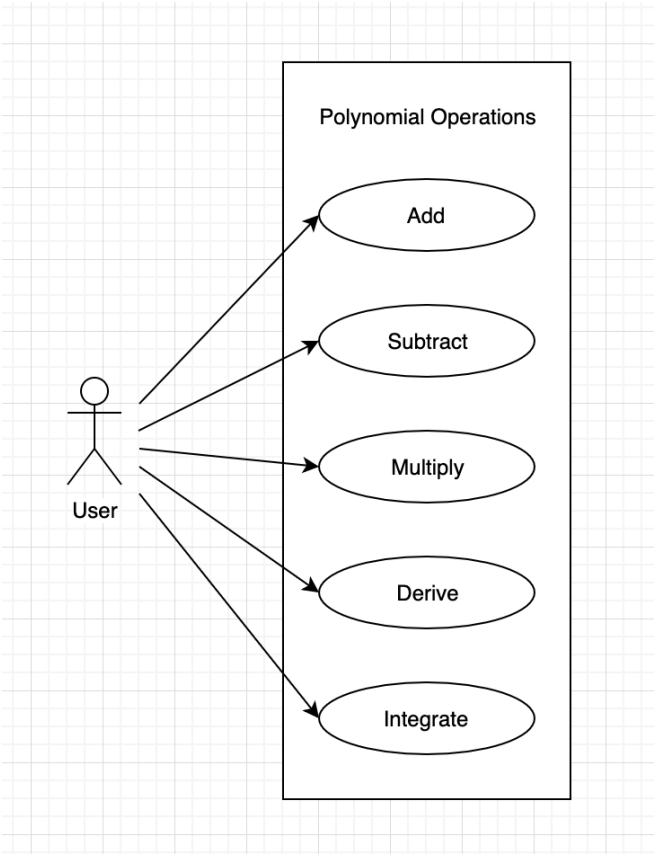
Secondary objectives:
- Creating a good looking, fully-functional graphical user interface.
- Implementing algorithms to perform operations on polynomials.
- Linking the user interface to the implemented algorithms, in order to read the input and display the output correctly to the user.

All the secondary objectives presented above will be detailed in the fourth chapter of this documentation.

2. Problem analysis, modelling, use-cases

The application is built to perform operations on polynomials of one variable and integer coefficients. It performs operations on two polynomials: addition, subtraction and multiplication, as well as operations on one polynomial: derivation and integration.
The integration operation will return a result polynomial with real coefficients.

To better illustrate how the application can be used, I have built a use-case diagram:



Use-cases description:

- **Addition**: The operation will be performed when the user presses the button “Add”. In order for the operation to work, the user needs to input two polynomials, otherwise a pop-up will appear to inform the user that the input is not valid. Next, the application will read the two polynomials and will use the “add” method to compute the result of addition, then the “toString” method to prepare the result polynomial for display (i.e. to build a nice-looking String that can be read and understood by the user). The last step is to place the resulted String in the “Operation Result” field, so that it is displayed on the graphical user interface.

- **Subtraction**: The operation will be performed when the user presses the “Subtract” button. In order for the operation to work, the user needs to input two polynomials, otherwise a pop-up will appear to inform the user that the input is not valid. Next, the application will read the two polynomials and will use the “subtract” method to compute the result of subtraction, then the “toString” method to prepare the result polynomial for display (i.e. to build a nice-looking String that can be read and understood by the user). The last step is to place the resulted String in the “Operation Result” field, so that it is displayed on the graphical user interface.

- **Multiplication**: The operation will be performed when the user presses the “Multiply” button. In order for the operation to work, the user needs to input two polynomials, otherwise a pop-up will appear to inform the user that the input is not valid. Next, the application will read the two polynomials and will use the “multiply” method to compute the result of multiplication, then the “toString” method to prepare the result polynomial for display (i.e. to build a nice-looking String that can be read and understood by the user). The last step is to place the resulted String in the “Operation Result” field, so that it is displayed on the graphical user interface.

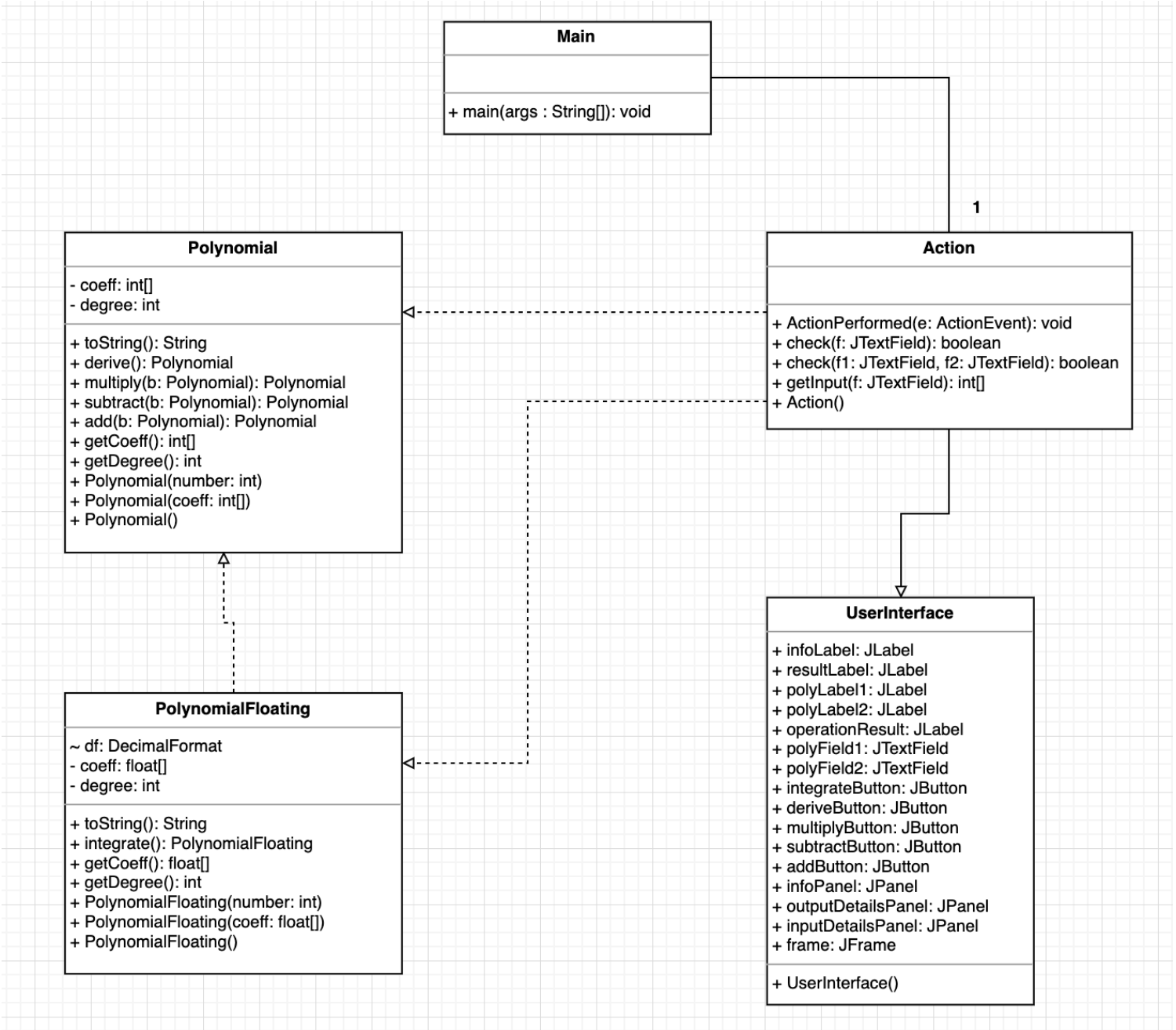
- **Derivation**: The operation will be performed when the user presses the “Derive” button. In order for the operation to work, the user needs to input a polynomial, otherwise a pop-up will appear to inform the user that the input is not valid. Next, the application will read the polynomial and will use the “derive” method to compute the result of derivation, then the “toString” method to prepare the result polynomial for display (i.e. to build a nice-looking String that can be read and understood by the user). The last step is to place the resulted String in the “Operation Result” field, so that it is displayed on the graphical user interface.

- **Integration**: The operation will be performed when the user presses the “Integrate” button. In order for the operation to work, the user needs to input a polynomial, otherwise a pop-up will appear to inform the user that the input is not valid. Next, the application will read the polynomial and will use the “integrate” method to compute the result of integration, then the “toString” method to prepare the result polynomial for display (i.e. to build a nice-looking String that can be read and understood by the user). The last step is to place the resulted String in the “Operation Result” field, so that it is displayed on the graphical user interface.

3. Design (Decisions, UML Diagrams, Class Design, User Interface)

The software design approach is OOP style. The application uses a system of interacting object for the purpose of solving the proposed problem.

The UML Diagram of the system is presented below:



The classes are illustrated as rectangles divided into 3 parts: the first one (up) contains the name of the class, the second (middle) contains the attributes and the third (down) contains the implemented methods.

The relationships between classes are represented with different types of links. The UserInterface class has an ‘inheritance’ type of relationship with the Action class, because the latter inherits the features of the UserInterface class. This type of relationship is illustrated with a continuous line with an arrow at the end.
There are multiple ‘dependency’ relationships drawn in the diagram. For example, the Action class is has a dependency type relationship with the Polynomial class, because, in order to perform certain actions defined in the Action class, objects of type Polynomial are needed. In order to illustrate this type of relationship, a dashed line with an arrow at the end was used.
Between Main and Action classes, there is an ‘Association’ type of relationship.

4. Implementation

The application consists of four packages containing a total of five classes, that reach all the secondary objectives mentioned in the first chapter of the documentation. These classes will be presented and detailed below.

• The Main class

This is a class used to run the application. It has only one method: main( ), in which the constructor of class Action is called.

• The Polynomial class

This class is used to represent polynomials that have integer coefficients. Each Polynomial object is described by its degree and coefficients array, and the implemented methods all operate on these two attributes. The important methods present in this class are:

- add: used to add the polynomial received as parameter to the polynomial the method is called on. It returns a new polynomial with the degree equal to the maximum between the degrees of the first and second polynomials, and the coefficients array corresponding to the sum of the two polynomials.
- subtract: used to subtract the polynomial received as parameter form the polynomial the method is called on, and store the result in a third polynomial. It does this in a similar way to the “add” method.
- multiply: method used to perform the multiplication operation on two polynomials: the one received as parameter and the one the method is called on. The result is stored in a third polynomial, that has a degree equal to the sum of degrees of the first and second polynomials.
- derive: this method has no parameters. It performs the derivation operation on the polynomial it is called on. The result is a new polynomial with the degree equal to the degree of the initial polynomial, minus one.
- toString: has no parameters. This method is used to build a string from the polynomial information (degree and coefficients array), in order to be displayed in the graphical user interface as the operation result. It is called on the polynomial object that is the result of an operation.

- **The PolynomialFloating class**

This class is used to represent polynomials that have real coefficients. Each PolynomialFloating object has a degree and a coefficients array as attributes.
The most important methods implemented in this class are:
- integrate: method with no parameters. Its purpose is to perform the integration operation on the polynomial object it is called on. The result is a new polynomial with the degree equal to the degree of the initial polynomial, plus one.
- toString: has no parameters. This method is called on a polynomial object, in order to create its string version (polynomials are represented only by degree and coefficients array). The result string will be displayed in the user interface, as the operation result. This method is very similar to the one implemented in the Polynomial class, only modified to display real coefficients with two decimals.

- **The UserInterface class**

It is used to build the graphical user interface of the application. This class is crucial for developing a good application, since it represents the median between the user and the designed system. It is what the user sees and uses in order to obtain the desired results.
The method implemented in this class is:
- UserInterface: this is a constructor method that builds the whole design of the application. It sets the dimensions of the frame, places the three panels on the frame, and all the corresponding elements of the design in the right panel. The whole design of the GUI is documented with line comments inside the UserInterface class of the application.

- **The Action class**

This is the class used to link the UserInterface class to the Polynomial and PolynomialFloating classes. All the buttons in the GUI have action listeners so that, when the user presses a button, the application will perform the desired operation. This class also contains methods that read and validate the input received from the user.
The most important methods that are implemented in this class are:
- Action: this is the constructor method of the class. It is very important, because this is where all the action listeners are added to the buttons in the user interface. Also, this is the only method called in the Main class, used to run the application.
- actionPerformed: has as parameter an action event. The method is used to select and perform the required action, based on the button pressed by the user: the input is read (one or both text fields, depending on the operation), validated (using the methods presented below) and then used as parameters for the operation. Finally, the result of the operation is printed nicely in the result field, so the user can see it.
- getInput: used to read the polynomials entered by the user on the input text fields. It also checks for symbols other than integers and displays a pop-up to inform the user that the provided polynomial is not valid, if needed.
- check: can have one or two parameters, depending on which operation was requested. It checks if the polynomials were entered into the text fields, and displays an error pop-up otherwise.

5. Results

In order to test the methods presented above, a JUnit test class was implemented. It is called “JUnitOperationsTest” and contains test methods for addition (testAdd), subtraction (testSubtract), multiplication (testMultiply), derivation (testDerive) and integration (testIntegrate).
The tests consist in creating one or two polynomials (depending on the operation), calling the corresponding method on the inputs and then comparing the result to an expected result, already set by the tester.

6. Conclusions

After developing this application, I feel like I have learned a lot. It is the first ‘complete’ application that I designed and implemented, having both a front-end and a back-end. It was a challenge that encouraged me to learn about new programming/design techniques and start applying them.

Future developments:

- Implement a division method
- Implement functionality that allows the user to view a 2D representation of the polynomial
- Implement a method that evaluates the polynomial for a certain value of x

7. Bibliography

- YouTube videos used to learn about the graphical user interface
https://www.youtube.com/watch?v=jUEOWVjnIR8&t=1s

- Oracle documentation on Java Swing
https://docs.oracle.com/javase/tutorial/uiswing/start/index.html

- JavaTPoint Swing tutorial
https://www.javatpoint.com/java-swing

- UML Diagrams theory and examples
https://tallyfy.com/uml-diagram/

- Lecture slides about UML diagrams, use-case diagrams and general design

- A considerable amount of Stack Overflow searches for solving bugs or unexpected errors.