DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: GRAD LAURENȚIU-CĂLIN
GROUP: 30425

Contents

1. Assignment Objective

The main objective of this assignment is to design and implement a polynomial calculator that can perform the following operations:

a) addition
b) subtraction
c) multiplication
d) division
e) derivation
f) integration

2. Problem Analysis, Modeling, Scenarios, Use Cases

The problem is quite straightforward: we must create some sort of application that performs arithmetic and functional operations over polynomials. There is no need to develop or choose between algorithms for any of those operations because the brute force solution is optimal. Some abstraction is necessary: in mathematics a polynomial is represented as $\sum_{k=0}^{n} a_k X^k$ where X is the unknown, a_k is the coefficient and k represents the power. This way of looking at a polynomial can be simplified drastically if we want to store it. Because all polynomials follow the same representation pattern, we can imagine them as vectors (arrays) of coefficients where the k^{th} element is the integer that multiplies X^k in the mathematical expression. This means we must store only the coefficients of each polynomial in an abstract data type.

The polynomial calculator is designed to perform the operations mentioned in the first chapter if the required number of polynomials is provided. The polynomials need not to be strongly typed (i.e. coefficient * X ^ power where coefficient and power are integers), but the order convention must be satisfied for correct functionality (for example, X^2*100 is not a valid format; it should be replaced with 100*X^2 or 100X^2). Being a simple application, there are not a lot of steps to be followed when one intends to use it:

a) Run the application: a pop-up window appearing means the application is ready to go. The window is composed of two parts: the part where the background is pink is where the polynomials should be inserted (the first two text boxes that are editable from top to bottom), and the part with the green background where buttons are placed. Each button performs the operation that its name indicates. If the input is wrong or does not respect the order convention a smaller pop-up window appears announcing the user that there is a problem regarding the input. Only if the input is correct does the computation begin.
b) The user inserts either one polynomial (if he intends to find the result of derivation or integration) or two (for any of the other operations). When subtracting/dividing the first polynomial (field from top to bottom) acts as the minuend/dividend and the second polynomial (field from top to bottom) acts as the subtrahend/divisor.
c) The user clicks the button corresponding to the operation he desires. The result is displayed in the third field in the part of the screen with the pink background.
d) By the time the result is displayed the calculator is ready to perform a new one, so the user is free to go back to step b)

The user diagram describes the dependencies between the user's interactions and the system the polynomial calculator is. The aim of the user is to compute the result of one of the mentioned operations, but the system does not let him do this unless the parameters are provided. At the same time, for enhanced development possibilities the functionality of the application is divided into two: the operations performed on a single polynomial and the operations that require two polynomials. These two are the extensions of the general functionality of the calculator. The current user represents the only human instance needed.

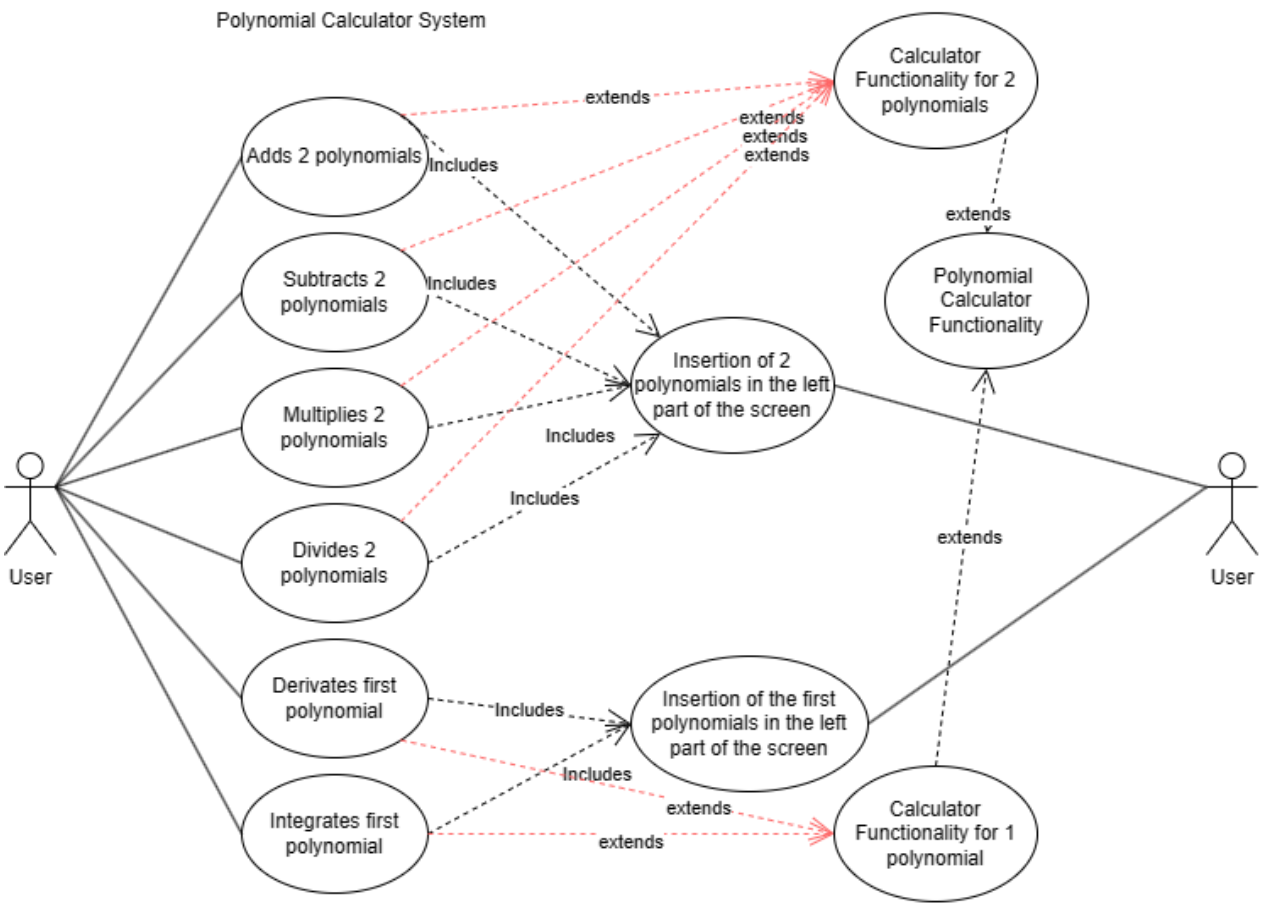


Figure 1. The use case diagram

3. Design

I have chosen the classical approach in solving projects that require graphical user interfaces: the calculator is divided conceptually into two main directories – backend and frontend. The backend is where the processing and the computation happen and everything that is related to the appearance is grouped in the frontend package.

The backend package consists of two abstract data types defined by me: String Polynomial and Polynomial. In addition to these there are two other classes that I regard as utility resources: Pair and Pattern. My implementation requires all these to convert the input polynomial into an array of coefficients.

The frontend package is divided based on a layout of slices. Each object in the window acts as a stamp that is applied over the previous layer. This means there are 2 main panels: the InputPanel (pink background) and the ControlPanel (green background). In the InputPanel there are 2 other pane-like components: a ReadPanel that consists of two fields where the polynomials are supposed to be inserted and an OutputPanel where the result of the computation is displayed. The purpose of the ControlPanel is to house the 6 buttons that perform the operations enumerated in the first section. Each button is associated to an ActionListener that manages the interaction of the user with the medium of the application.

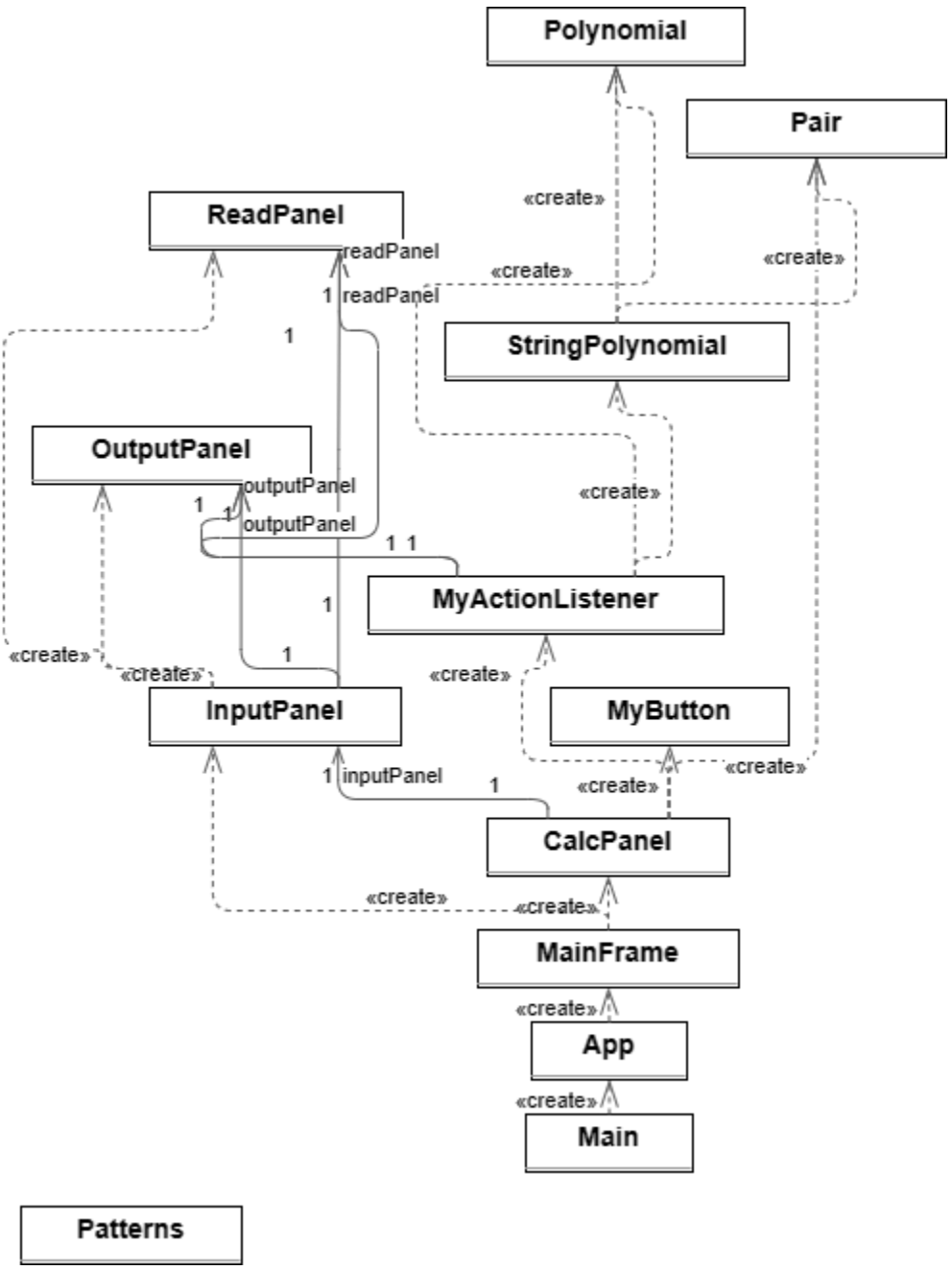
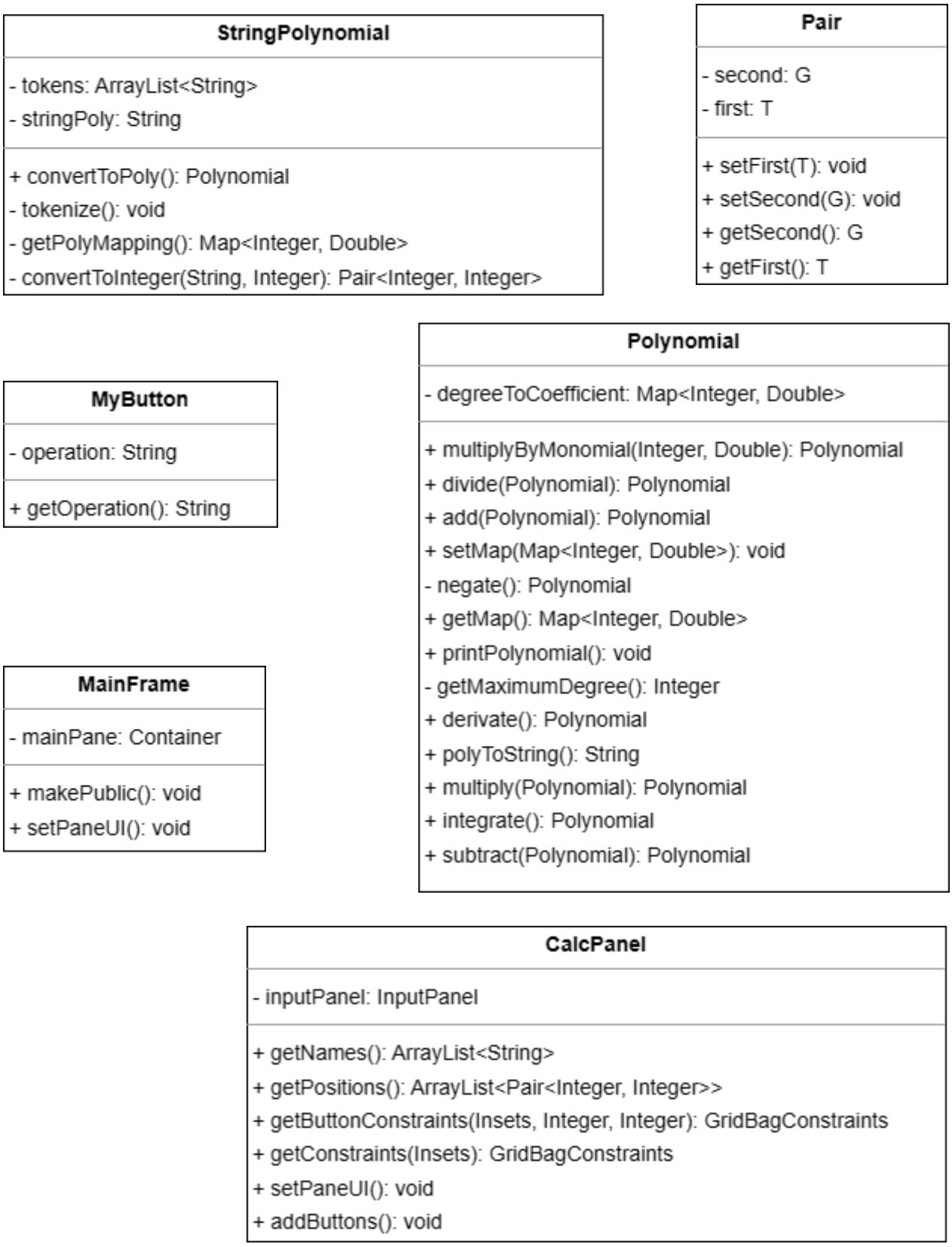
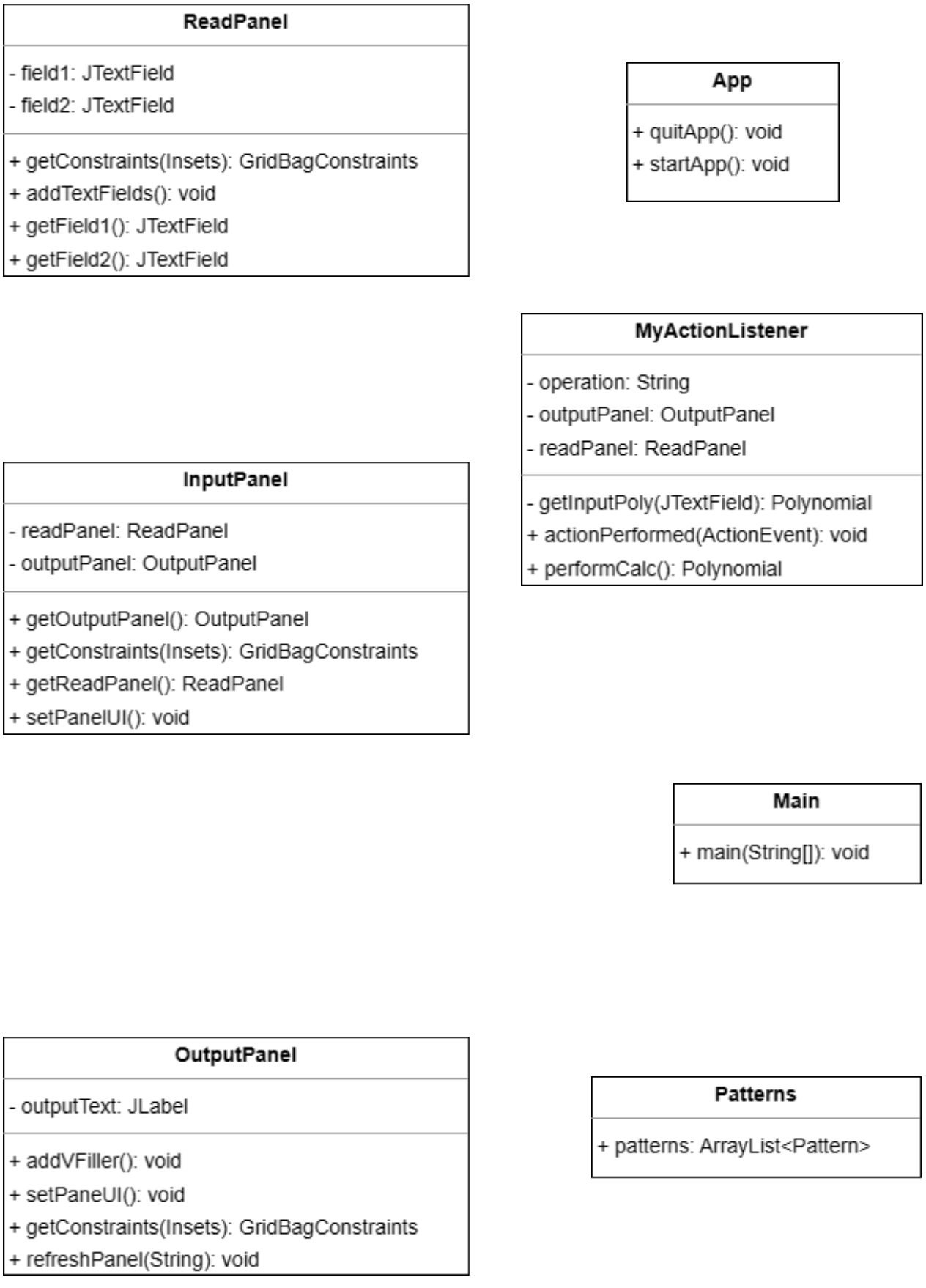


Figure 2. The UML diagram of the application.


StringPolynomial
- tokens: ArrayList<String>
- stringPoly: String
+ convertToPoly(): Polynomial
- tokenize(): void
- getPolyMapping(): Map<Integer, Double>
- convertToInteger(String, Integer): Pair<Integer, Integer>
Pair
- second: G
- first: T
+ setFirst(T): void
+ setSecond(G): void
+ getSecond(): G
+ getFirst(): T
MyButton
- operation: String
+ getOperation(): String
Polynomial
- degreeToCoefficient: Map<Integer, Double>
+ multiplyByMonomial(Integer, Double): Polynomial
+ divide(Polynomial): Polynomial
+ add(Polynomial): Polynomial
+ setMap(Map<Integer, Double>): void
- negate(): Polynomial
+ getMap(): Map<Integer, Double>
+ printPolynomial(): void
- getMaximumDegree(): Integer
+ derivate(): Polynomial
+ polyToString(): String
+ multiply(Polynomial): Polynomial
+ integrate(): Polynomial
+ subtract(Polynomial): Polynomial
MainFrame
- mainPane: Container
+ makePublic(): void
+ setPaneUI(): void
CalcPanel
- inputPanel: InputPanel
+ getNames(): ArrayList<String>
+ getPositions(): ArrayList<Pair<Integer, Integer>>
+ getButtonConstraints(Insets, Integer, Integer): GridBagConstraints
+ getConstraints(Insets): GridBagConstraints
+ setPaneUI(): void
+ addButtons(): void


ReadPanel
- field1: JTextField
- field2: JTextField
+ getConstraints(Insets): GridBagConstraints
+ addTextFields(): void
+ getField1(): JTextField
+ getField2(): JTextField
App
+ quitApp(): void
+ startApp(): void
MyActionListener
- operation: String
- outputPanel: OutputPanel
- readPanel: ReadPanel
- getInputPoly(JTextField): Polynomial
+ actionPerformed(ActionEvent): void
+ performCalc(): Polynomial
InputPanel
- readPanel: ReadPanel
- outputPanel: OutputPanel
+ getOutputPanel(): OutputPanel
+ getConstraints(Insets): GridBagConstraints
+ getReadPanel(): ReadPanel
+ setPanelUI(): void
Main
+ main(String[]): void
OutputPanel
- outputText: JLabel
+ addVFiller(): void
+ setPaneUI(): void
+ getConstraints(Insets): GridBagConstraints
+ refreshPanel(String): void
Patterns
+ patterns: ArrayList<Pattern>

4. Implementation

The implementation of the classes and of the main methods is discussed:

a) StringPolynomial

This class acts as a wrapper for the input data. A polynomial inserted in a JTextField is stored as a string in memory. The purpose of the StringPolynomial class is to store this string representation and by means of methods to decompose it into monomials (tokens) that are then processed until a corresponding pair (degree, coefficient) is found. Such a pair is inserted into a Map.

Important methods:

- **.tokenize()** converts a string into an array of monomials based on pattern matching (patterns are provided by the Patterns class). It also checks for the integrity of data: only polynomials that respect the canonical representations are accepted, however there are cases where the validation algorithm fails (X200 becomes X+200 instead of 200*X). My approach to tokenizing the string is to start from the most specific pattern and to decrease its accuracy step-by-step (i.e. from 1*X^1 to X). If characters are left, the string is rejected, and a pop-up window is displayed.
- **.getPolyMapping()** is the most important method of the class because it converts the array of tokens (monomials) into a map of degrees – coefficients. This process is performed on each token: its sign is defined as 1 (positive), if a minus is encountered it becomes -1, then the coefficient is processed and converted into an Integer by means of another method. The unknown (X) is jumped, as well as the '^' character. The exponent of the unknown is processed the same way as the coefficient.
- **.convertToPoly()** calls the above-mentioned methods and generates a Polynomial-class object

b) Polynomial

The main object that is the parameter of all the operations.

Important methods:

- **.add(Polynomial polynomial)** performs the addition of two polynomials. The result is stored in a new Polynomial and returned.
- **.subtract(Polynomial polynomial)** subtracts polynomial from this and returns the result. The operation calls a helper method that generates the negation of the second polynomial, then it is added to the object subtract is called on.
- **.multiply(Polynomial polynomial)** generates a new polynomial that is the result of the multiplication of this and the Polynomial parameter. The way this is done is the basic, brute-force, for-in-for algorithm.
- **.division(Polynomial polynomial)** returns the quotient of the division of this to the Polynomial parameter. The algorithm is the classical one and calls a .multiply method that multiplies a monomial with a Polynomial object together with the .subtract method.
- **.derivate()** performs the derivation term by term and returns the result.
- **.integrate()** performs the integration term by term and returns the result.

c) Patterns

This class is used to store the RegEx that I am looking for in the string.

No methods are instantiated here.

d) Pair

A utility class that uses 2 generics as parameters. Stores pairs of (Integer, Double) in most cases.

Only setters and getters.

e) MainFrame

The underlying canvas of all the GUI. All panels are stamped on top of it.

The methods set up the basic functionalities: when exited the application stops as well, the window is not resizable and is displayed. **.setPaneUI()** places the two panels where objects that support interaction with the user are placed.

f) InputPanel

Acts as a frame for other two panels.

The methods set up the size and place the ReadPanel and the OutputPanel inside.

g) ReadPanel

It is composed of two JTextFields that are used for inserting polynomials as input.

The methods specify the placement. Some getters are required.

h) OutputPanel

Its main purpose is to house the JTextArea that serves as an output field for the resulting computation.

The methods set the appearance and position inside the InputPanel. A specific method **.refreshPanel()** updates the output value based on the computational result.

i) CalcPanel

This panel houses the buttons that correspond to the mathematical operations that can be performed on polynomials: addition, subtraction, multiplication, division, derivation, and integration.

The methods place the buttons in a symmetric manner based on a GridBagLayout and a vector of coordinates.

j) MyActionListener

The class implements the ActionListener interface and makes the connection between user's interaction and the feedback of the application and the GUI. It stores references to the OutputPanel, the ReadPanel and the operation corresponding to the button.

Important methods:

- **.getInputPoly(JTextField textField)** is called when a button is pressed. It also sends a control to the OutputPanel to signal it to update.
- **.performCalc()** is the method that decides which button has been pressed and what operation should be performed. Depending on that either one or both JTextFields are saved and converted into Polynomial-type objects. The requested operation is performed afterwards.

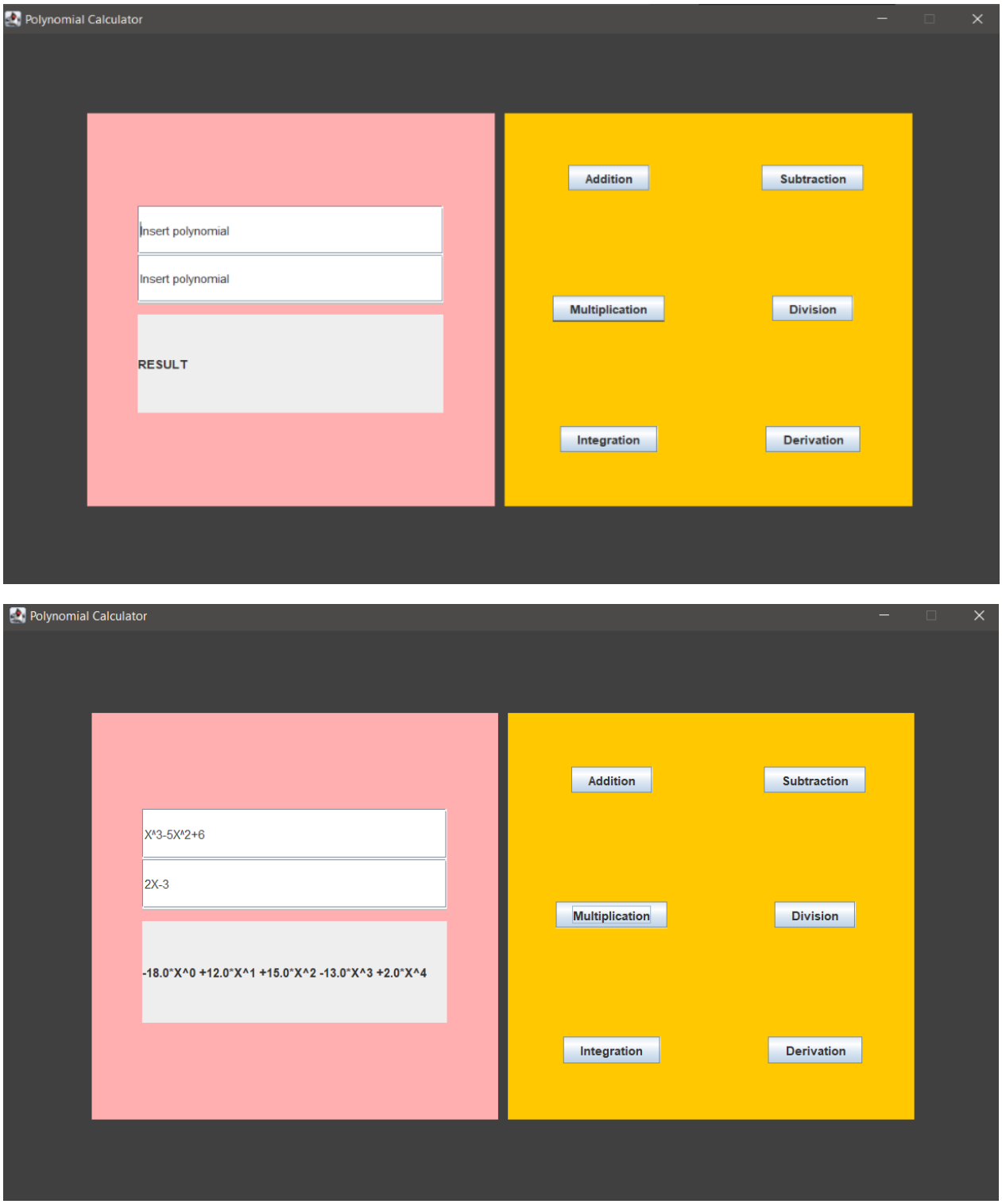


Figure 3. Views of the application

5. Results

All the mathematical functionalities have been tested and compared to human-computed results, double-checked by means of Microsoft web calculator. I aimed to take into consideration all critical cases such as multiplying by 0, integrating 0, dividing by 0 etc. I did not notice any erroneous computations for additions, subtractions, multiplications, integrations, or derivations. The only error I discovered in testing is caused by division, but I believe it is insignificant: when dividing polynomials whose division of coefficients generates irrational numbers (i.e. $\frac{2}{3} = 0.66666\ldots$) the representation in memory is, of course, truncated. This means the more operations the computer performs with this type of number the bigger the error becomes. In my test, after about three divisions and three multiplications (involving division by 13, 169 etc.) the error appeared to be in the range of ±3 in the order of the 16th digit (computationally speaking, this is a great and irrelevant approximation).

Test results (performed in JUnit):

a) Addition
   i. Test 1
      A. Test parameters: $P[X] = X^2 + X + 1$ and $Q[X] = -3X + 8$
      B. Correct result: $R[X] = P[X] + Q[X] = Q[X] + P[X] = X^2 - 2X + 9$
      C. Test result: passed.
   ii. Test 2
      A. Test parameters: $P[X] = -X^7 + 100X^4 + 50X^3 - 23X - 5$ and $Q[X] = X^7 - 100X^4 - 50X^3 + 23X + 5$
      B. Correct result: $R[X] = P[X] + Q[X] = Q[X] + P[X] = 0$
      C. Test result: passed.

b) Subtraction
   i. Test 1
      A. Test parameters: $P[X] = X^5 - X^3 + 30X - 5$ and $Q[X] = 7X^2 - 52$
      B. Correct result: $R[X] = P[X] - Q[X] = X^5 - X^3 - 7X^2 + 30X + 47$
      C. Correct result: $R[X] = Q[X] - P[X] = -X^5 + X^3 + 7X^2 - 30X - 47$
      D. Test result: passed.
   ii. Test 2
      A. Test parameters: $P[X] = 100X^{999} - 72X^{300} + 30X - 50$ and $Q[X] = 1201X^{999} - 31X^{300} + 30X + 20$
      B. Correct result: $R[X] = P[X] - Q[X] = -1101X^{999} - 41X^{300} - 70$
      C. Correct result: $R[X] = Q[X] - P[X] = 1101X^{999} + 41X^{300} + 70$
      D. Test result: passed.
   iii. Test 3
      A. Test parameters: $P[X] = X + 12$ and $Q[X] = X + 12$
      B. Correct result: $R[X] = P[X] - Q[X] = Q[X] - P[X] = 0$
      C. Test result: passed.

c) Multiplication
   i. Test 1
      A. Test parameters: $P[X] = 10X^5 + 3X^4 - 2X^3 + X^2 - 41X + 3$ and $Q[X] = 19X^6 - 3X^5 + 12X^3 - 12X^2 - X - 9$
      B. Correct result: $R[X] = P[X] * Q[X] = Q[X] * P[X] = 190X^{11} + 27X^{10} - 47X^9 + 145X^8 - 866X^7 + 110X^6 - 66X^5 - 529X^4 + 545X^3 - 4X^2 + 366X - 27$
      C. Test result: passed.

ii. Test 2
   A. Test parameters: $P[X] = X + 1$ and $Q[X] = X + 9$
   B. Correct result: $R[X] = P[X] * Q[X] = Q[X] * P[X] = X^2 + 10X + 9$
   C. Test result: passed.

iii. Test 3
   A. Test parameters: $P[X] = X + 1$ and $Q[X] = 0$
   B. Correct result: $R[X] = P[X] * Q[X] = Q[X] * P[X] = 0$
   C. Test result: passed.

d) Division

i. Test 1
   A. Test parameters: $P[X] = X + 1$ and $Q[X] = 0$
   B. Correct result: $R[X] = \frac{P[X]}{Q[X]} = 0$, impossible indeed, but I adopted 0 as a convention.
   C. Correct result: $R[X] = \frac{Q[X]}{P[X]} = 0$
   D. Test result: passed.

ii. Test 2
   A. Test parameters: $P[X] = X^3 + 3X^2 + 3X + 1$ and $Q[X] = X + 1$
   B. Correct result: $R[X] = \frac{P[X]}{Q[X]} = X^2 + 2X + 1$
   C. Correct result: $R[X] = \frac{Q[X]}{P[X]} = 0$
   D. Test result: passed.

iii. Test 3
   A. Test parameters: $P[X] = X^5 + 3X^3 - 3X^2 + 5X - 5$ and $Q[X] = X^2 + 7$
   B. Correct result: $R[X] = \frac{P[X]}{Q[X]} = X^3 - 4X - 3$
   C. Correct result: $R[X] = \frac{Q[X]}{P[X]} = 0$
   D. Test result: passed.

iv. Test 4
   A. Test parameters: $P[X] = 3X^7 - 3X^5 + 10X^3 + 5X^2 - 50$ and $Q[X] = 2X^3 + 4$
   B. Correct result: $R[X] = \frac{P[X]}{Q[X]} = \frac{3}{2}X^4 - \frac{3}{2}X^2 - 3X + 5$
   C. Correct result: $R[X] = \frac{Q[X]}{P[X]} = 0$
   D. Test result: passed.

v. Test 5
   A. Test parameters: $P[X] = 11X^3 + 7X^2 - 5X + 3$ and $Q[X] = 13X + 23$
   B. Correct result: $R[X] = \frac{P[X]}{Q[X]} = \frac{11}{13}X^2 - \frac{162}{169}X + \frac{2881}{2197}$
   C. Correct result: $R[X] = \frac{Q[X]}{P[X]} = 0$
   D. Test result: partially passed, the conversions from fractional to decimal generate errors (±3) at the **16th decimal**.

e) Derivation

i. Test 1
   A. Test parameters: $P[X] = 11X^3 + 7X^2 - 5X + 3$
   B. Correct result: $R[X] = \frac{d}{dX}P[X] = 33X^2 + 14X - 5$
   C. Test result: passed.

ii. Test 2

A. Test parameters: $P[X] = 0$

B. Correct result: $R[X] = \frac{d}{dX} P[X] = 0$

C. Test result: passed.

f) Integration

i. Test 1

A. Test parameters: $P[X] = X^2 + 5X + 3$

B. Correct result: $R[X] = \int P[X] dX = \frac{1}{3} X^3 + \frac{5}{2} X^2 + 3X$

C. Test result: passed.

ii. Test 2

A. Test parameters: $P[X] = -9X^8 - 5X^4 + 11$

B. Correct result: $R[X] = \int P[X] dX = -X^9 - X^5 + 11X$

C. Test result: passed.

6. Conclusions

Working on this assignment felt great after two weeks of not keeping in touch with Java. The task itself was not that complex, but there were a lot of small things to keep track of. The real challenge was to break down the calculator itself into independent modules that can perform more general tasks or are (more or less) open to extension while keeping it simple and clear. I did not have a lot of fun while creating the graphical user interface, but, in the end, I feel like I am more aware of what layouts can do, how action listeners are supposed to be designed or how frames and panels interact. I practiced the decomposition of classes into subclasses and aimed the design short and effective methods that are more Java-like than C.

The final application is not the best it can be for sure. I consider its main defects to be the graphical user interface and the input-output polynomial representation. The GUI is simple and does its job, but it does not make anyone want to try or spend more time than needed with the application running. Its roughness is not a point of focus for a software engineer, but if it were to be used by any other people but engineers, no one would consider the way it looks appealing. It does compute correctly, though. At the same time, I hate the way I must insert polynomials as strings because I feel like I lose a lot of time searching for ^ or pressing shift + x for a capital letter. This input method should be optimized. The way the result is shown at output is not pleasant, as well. It seems too hard to follow even for me.

Other further developments:

- Improved GUI.
- Improved class design.
- Improved and simplified inter-class connections.
- Improved, simplified, and more specialized intra-class connections.
- New operations: negation & multiplying by a monomial (already implemented as private for subtraction), rising to a power, evaluation at a certain X value.
- Perfectioning the division operation so that it prints a polynomial in fractional form rather than decimal for superior precision.
- Cleaner, more explicit code style.

7. Bibliography

1. Unified Modeling Language - Wikipedia
2. Use Case Diagrams | Unified Modeling Language (UML) - GeeksforGeeks
3. aa.png - draw.io (diagrams.net)
4. Use Case Diagram Tutorial (Guide with Examples) | Creately
5. Index of /courses/pt/materials/lectures (dsrl.eu)
6. RegExr: Learn, Build, & Test RegEx
7. regex101: build, test, and debug regex
8. Regular expression - Wikipedia
9. java - make a JLabel wrap it's text by setting a max width - Stack Overflow
10. A Guide to JUnit 5 | Baeldung
11. Polynomial Equation Calculator (symbolab.com)
12. A Visual Guide to Layout Managers (The Java™ Tutorials > Creating a GUI With Swing > Laying Out Components Within a Container) (oracle.com)
13. class.png - draw.io (diagrams.net)
14. Examples | Microsoft Math Solver
15. Programming Techniques lectures, laboratories, and support materials
16. Paul Deitel, Harvey Deitel, Java How to program (10th edition), Publisher: Pearson Education, Inc., Upper Saddle River, NJ, United States, ISBN: 978-0-13-380780-6, Published: 2015