DOCUMENTATION ASSIGNMENT 1

STUDENT NAME: BARTHA TUDOR

GROUP:30425

CONTENTS

1.Assignment Objective

Main Goal: Develop a user-friendly graphical interface for a polynomial calculator capable of performing addition, subtraction, multiplication, division, derivative, and integration operations on polynomials with single-variable and integer coefficients.

Sub-Objectives:

1. **Design Interface**
2. **Implement Polynomial Input**
3. **Implement Operations**
4. **Integrate Interface with Operations**
5. **Handle Errors**
6. **Test and Debug**
7. **Optimize Performance**
8. **Documentation**

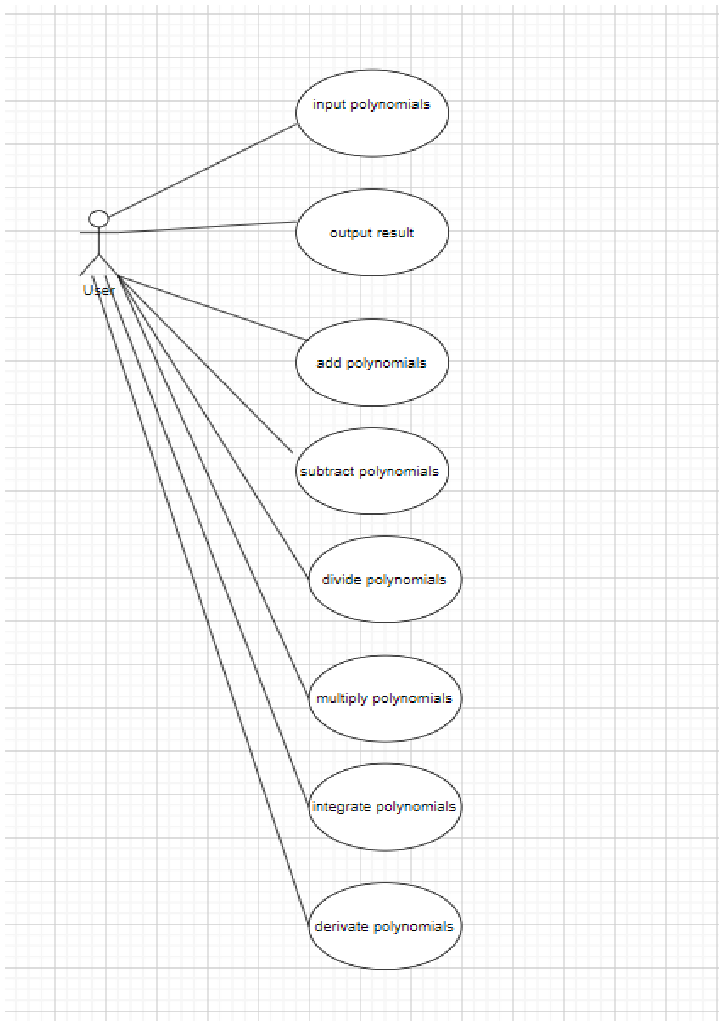
2. Problem Analysis, Modeling, Scenarios, Use Cases

This project at first sight looks fairly simple. However, at a second glance we realize that the actual challenges are to be able to implement the calculator bug-free. In mathematics, we don't pay attention to particular cases such as division by 0; we all know that is impossible, and we expect most people we interact with know this. We must take into consideration that this app might be used by inexperienced users. This was only one example to underscore the subtle complexity of the project. With this said, I will go into more detail about the abstraction process.

Functional requirements:

- **Polynomial Input:** Users should be able to input polynomials with single-variable and integer coefficients.
- **Mathematical Operations:** The calculator should support addition, subtraction, multiplication, division, derivative, and integration operations on polynomials.
- **User Interface:** The graphical interface should be intuitive, allowing users to easily input polynomials, select operations, and view results.
- **Error Handling:** The system should handle errors gracefully, providing clear messages for scenarios like division by zero or invalid input polynomials.

Use Case diagram:



Input Polynomial:

- **Actor:** User
- **Description:** User inputs a polynomial into the calculator.

Firstly, a pop-up window will appear, the GUI. The user writes in the two text fields destined for input 2 polynomials, or one depending on what operation he chooses to do. In the case of derivation and integration, only the first polynomial will be considered. The polynomials must be inputted in the following form: coefficient*x^exponent+ or -, you can also input parameters without coefficient if it is 1. The system stores the text for further processing.

Perform Mathematical Operation:

- **Actor:** User
- **Description:** User selects a mathematical operation to perform on the input polynomials.

This step will be executed properly only if valid input is given. The user selects the desired mathematical operation (addition, subtraction, multiplication, division, derivative, or integration). The selection is done by pressing one of the buttons from the GUI.

View Result:

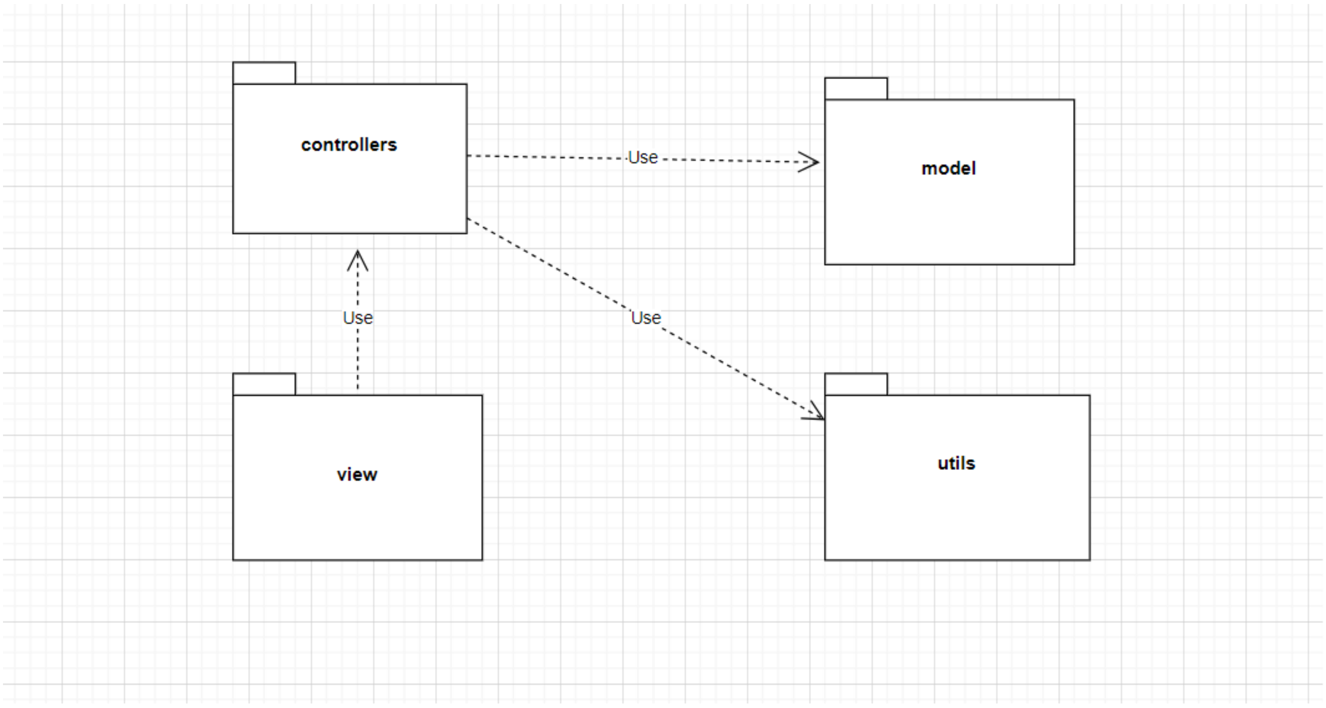
- **Actor:** User

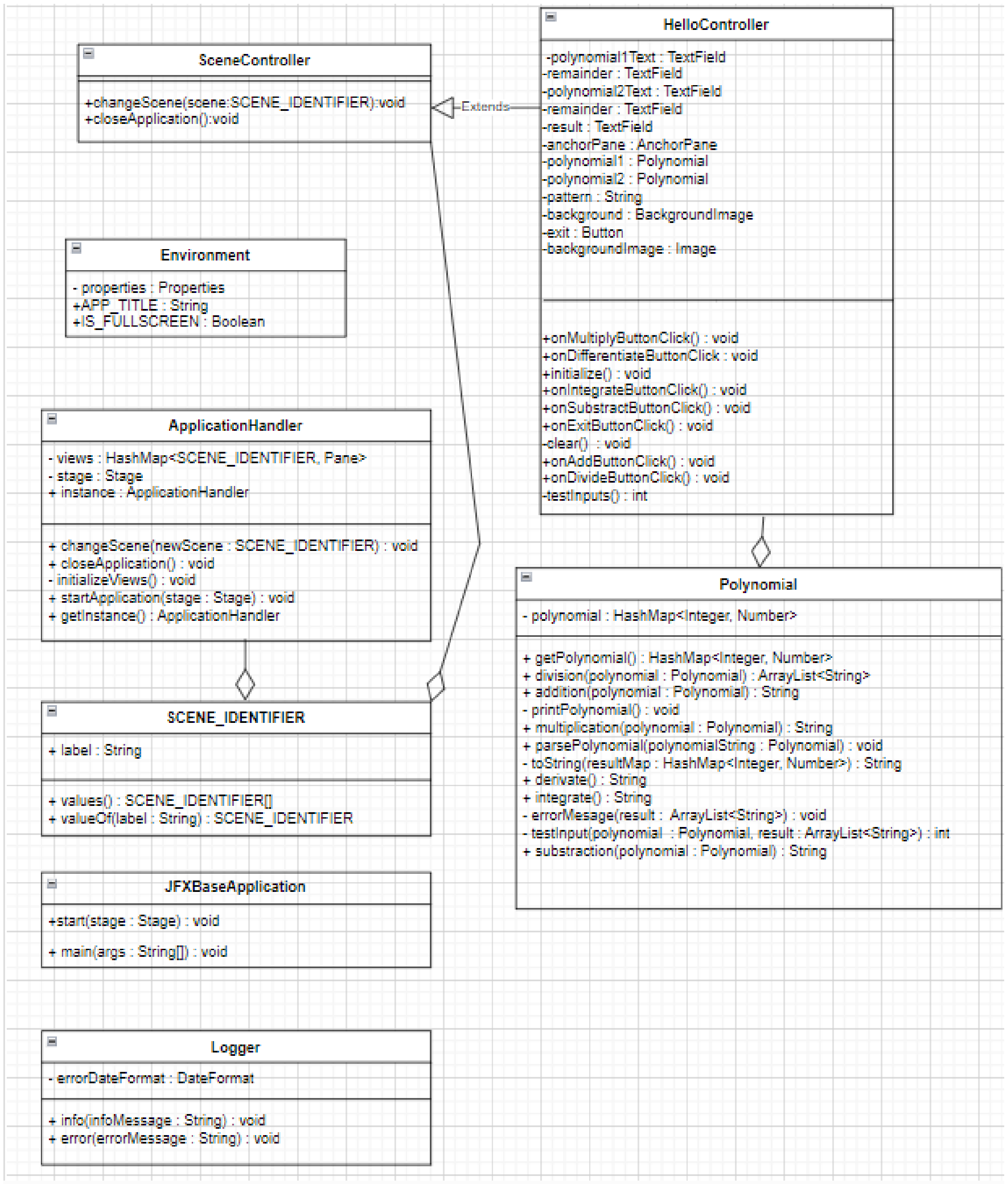
- **Description:** User views the result of the performed mathematical operation.
  The result will pop in the results field from the GUI. In the case of the division there are 2 different fields, one for quotient and one for remainder, else the results appear in the first field.

3. Design

I used an OOP approach to the problem, defining a Polynomial class was the first thing that came to mind. Here I used HashMap structure to store the coefficients and exponents of the polynomial. This was the best structure from a memory point of view, as using an ArrayList would have been just as fast, but it would have taken more memory. I implemented all the operations inside the Polynomial class as methods. Also, I did not see the need for a separate Monomial class, so I chose to implement the modularity inside the class, having multiple methods. I used JavaFX for the frontend, so the rest of the classes are related to the GUI. I also have controller classes; these are part of the controller package. Another package is the model package-this contains the Polynomial. And finally, I have the fxml file and the .css file used for modeling the GUI. Regarding the algorithms, the only algorithm I could say I used is the long division algorithm. The rest of the operations were straight forward.

Now I will present the package diagram and class diagram.




SceneController
+changeScene(scene:SCENE_IDENTIFIER):void
+closeApplication():void
Extends
HelloController
-polynomial1Text : TextField
-remainder : TextField
-polynomial2Text : TextField
-remainder : TextField
-result : TextField
-anchorPane : AnchorPane
-polynomial1 : Polynomial
-polynomial2 : Polynomial
-pattern : String
-background : BackgroundImage
-exit : Button
-backgroundImage : Image
+onMultiplyButtonClick() : void
+onDifferentiateButtonClick : void
+initialize() : void
+onIntegrateButtonClick() : void
+onSubstractButtonClick() : void
+onExitButtonClick() : void
-clear() : void
+onAddButtonClick() : void
+onDivideButtonClick() : void
-testInputs() : int
Environment
- properties : Properties
+APP_TITLE : String
+IS_FULLSCREEN : Boolean
ApplicationHandler
- views : HashMap<SCENE_IDENTIFIER, Pane>
- stage : Stage
+ instance : ApplicationHandler
+ changeScene(newScene : SCENE_IDENTIFIER) : void
+ closeApplication() : void
- initializeViews() : void
+ startApplication(stage : Stage) : void
+ getInstance() : ApplicationHandler
Polynomial
- polynomial : HashMap<Integer, Number>
+ getPolynomial() : HashMap<Integer, Number>
+ division(polynomial : Polynomial) : ArrayList<String>
+ addition(polynomial : Polynomial) : String
- printPolynomial() : void
+ multiplication(polynomial : Polynomial) : String
+ parsePolynomial(polynomialString : Polynomial) : void
- toString(resultMap : HashMap<Integer, Number>) : String
+ derivate() : String
+ integrate() : String
- errorMesage(result : ArrayList<String>) : void
- testInput(polynomial : Polynomial, result : ArrayList<String>) : int
+ substraction(polynomial : Polynomial) : String
SCENE_IDENTIFIER
+ label : String
+ values() : SCENE_IDENTIFIER[]
+ valueOf(label : String) : SCENE_IDENTIFIER
JFXBaseApplication
+start(stage : Stage) : void
+ main(args : String[]) : void
Logger
- errorDateFormat : DateFormat
+ info(infoMessage : String) : void
+ error(errorMessage : String) : void

4. Implementation

- **Polynomial**

I will start with this class, as it is the main class of the project. The only field I use is the polynomial which is a HashMap<Integer, Number>.I used the Number class to be able to represent the coefficients as Integer or Double.

.parsePolynomial(polynomialString : String)-This method takes a string as a parameter and then using regex manages to separate the terms and take the coefficient and exponent and turn them into a value-key group in the polynomial HashMap.I used a Matcher object and a Pattern object for tokenizing the String and being able to separate terms.

.toString(resultMap : HashMap<Integer, Number>)-This method transforms the resultMap into a String using iterators to iterate through the map. Based on the coefficient stored in the map I will put + or – in the StringBuilder object I use. Then I will put the coefficient-int or double then '*', then "x^" if the exponent is different from 0, and finally the exponent. This method is used only internally to directly pass a String to the controller from the mathematical methods.

.addition(polynomial : Polynomial)-This method is used for the addition of two polynomials: the one given as a parameter and the one which calls the method. The method is simple: adds the values from the 2 maps, if one map does not contain a method that the other map contains, then we will add 0.

.subtraction(polynomial : Polynomial) -This method is used for the subtraction of two polynomials: the one given as a parameter and the one which calls the method. The method is simple: subtract the values from the 2 maps, if one map does not contain a method that the other map contains, then the missing values will be 0.

.multiplication(polynomial : Polynomial) -This method is used for the multiplication of two polynomials: the one given as a parameter and the one which calls the method. The method is simple: we multiply every value and key from the hash map with every value and key from the other hash map. The algorithm is simple :a classic for in for. If a slot in the result hash map already has something in it, then we add the existing coefficient.

.division(polynomial : Polynomial)-This was the most complex operation to implement. Thankfully, the long division algorithm was presented at the laboratory, so I took that pseudocode and implemented it in java. I have two hash maps: quotient which is initially empty and the remainder which is polynomial HashMap that calls the method. This two maps will be converted to strings using the toString() method and will be passed to the controller as an ArrayList.

.derivate()-This method was very simple, it works on a single polynomial. The approach was to go through every key-value slots and perform derivation: multiplying the coefficient with the exponent-that will be the new coefficient and then subtracting 1 from the exponent.

.integrate()-This method was also simple, it works on a single polynomial. The approach was to go through every key-value slots and perform integration: dividing the coefficient with the exponent +1-that will be the new coefficient and then adding 1 to the exponent.

testInput and erorMessage-these are two private methods that validate the input,checking for division by 0 is their purpose

- **SCENE_IDENTIFIER**

This is an enum that is used to identify the different scenes from the project. In our case we only have one scene.

- **Logger**

This class is used to output to the console any infos or errors that may appear during the starting of the application and helps the developer identify the problem much easier: it shows exactly which scene is problematic, or gives an info stating that there were no problems during the initialization phase.

- **Environment**

This Java class, is responsible for loading environment-specific properties from a properties file (**1.properties**) and making them accessible to other parts of the application.

    - **APP_TITLE**: Represents the title of the application, which is retrieved from the properties file.
    - **IS_FULLSCREEN**: Represents a boolean flag indicating whether the application should run in fullscreen mode, also retrieved from the properties file.

- **ApplicationHandler**

This is the class that “runs” the whole project.

.startApplication(stage : Stage)-This method starts the application and defines the main aspects of how the GUI looks and behaves.

.closeApplication()-This methods defines the way the app behaves when we want to exit it. A pop-up will appear

.initializeViews()-This method check that every scene we may access in the project is correct and no errors may appear during the running of the app. It will point to the fxml file that has problems.

- **SceneController**

This class is the parent of all controllers .It implements basic functions: change scene and closeApplication.

- **HelloController**

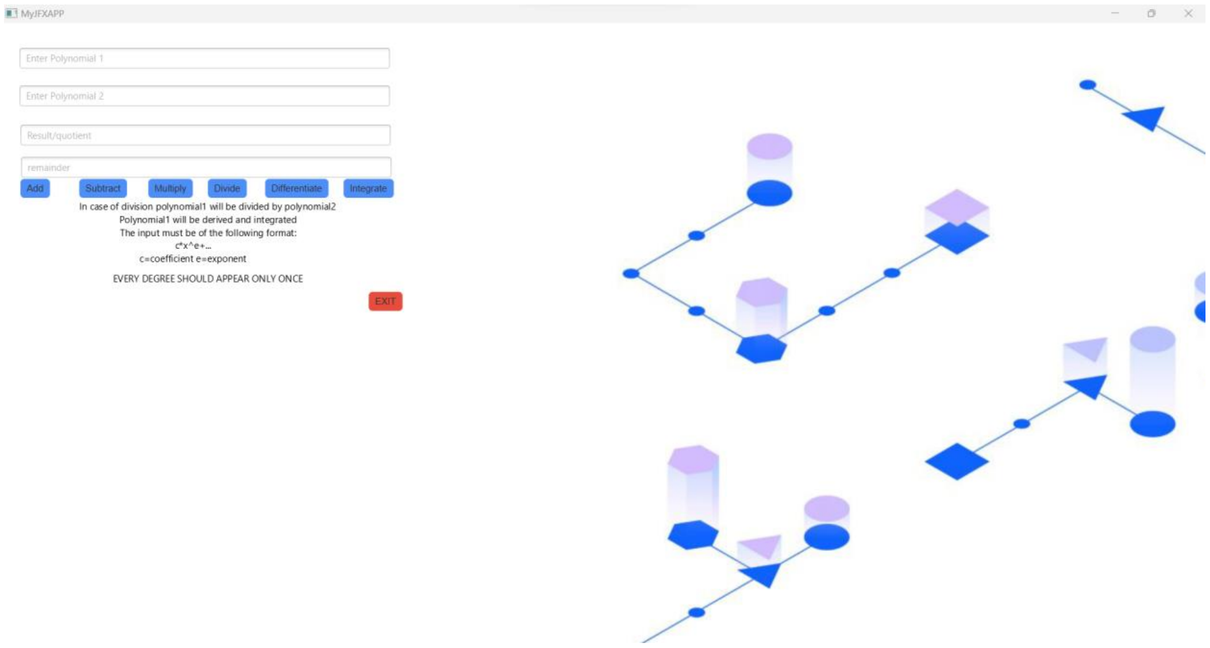
This class controls the only scene we have. The background is set in this class. There is a function testInputs that tests for the validity of the inputs, this is a supplementary check ,separate from the one done in the parsePolynomial(). Besides this there are only simple methods, they all look the same , and each performs an operation. This methods make the connection between the frontend and the backend of the project.

- **PolynomialTest**

This class is my JUnit testing class and it performs 10 tests on each the addition and the subtraction methods from the Polynomial class.

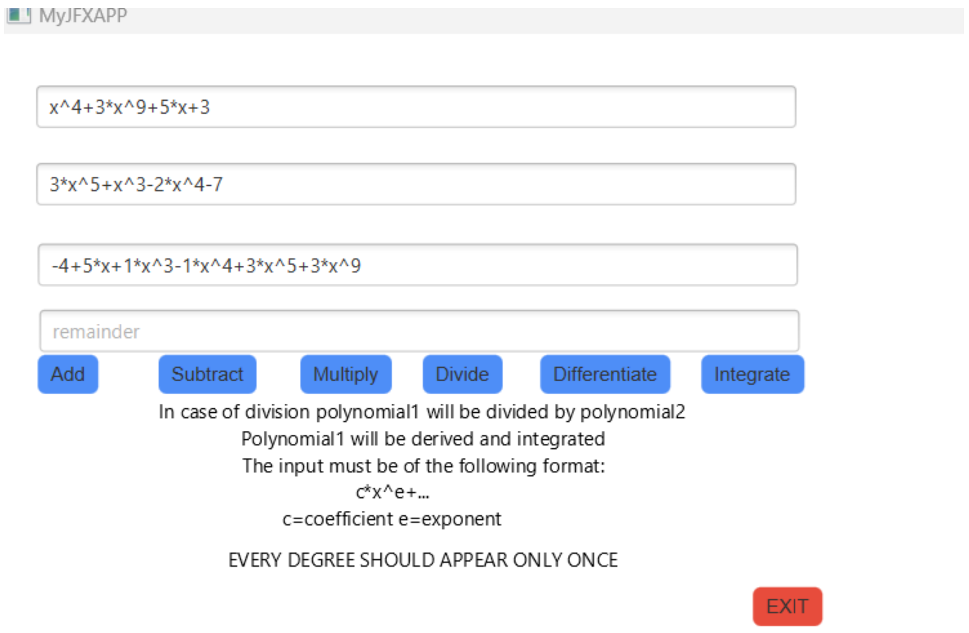
For the graphical user interface I used JavaFX. The modeling of the GUI is done using the hello-view.xml which defines the buttons, the anchor pane ,the text fields and the texts used. Also, each button is assigned a method from the HelloController.

I opted to use CSS styling; the file hello-view.css defines the colors of the buttons and various aspects of the GUI.



In case of division polynomial1 will be divided by polynomial2
Polynomial1 will be derived and integrated
The input must be of the following format:
c*x^e+...
c=coefficient e=exponent

EVERY DEGREE SHOULD APPEAR ONLY ONCE



x^4+3*x^9+5*x+3

3*x^5+x^3-2*x^4-7

-4+5*x+1*x^3-1*x^4+3*x^5+3*x^9

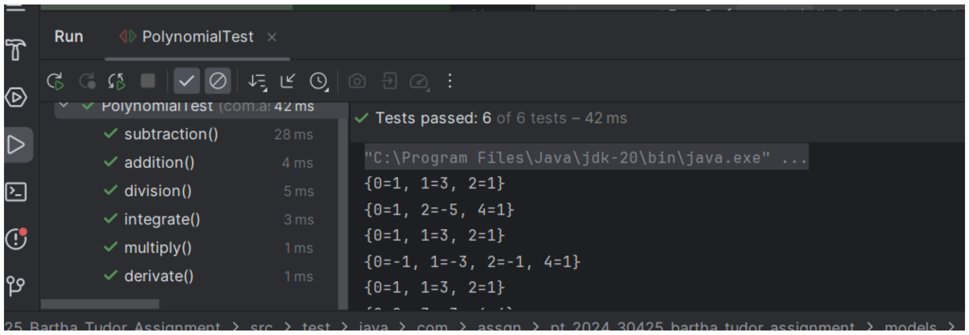
In case of division polynomial1 will be divided by polynomial2
Polynomial1 will be derived and integrated
The input must be of the following format:
c*x^e+...
c=coefficient e=exponent

EVERY DEGREE SHOULD APPEAR ONLY ONCE

5. Results

The operations all work, as I tested them manually and discovered an error in my division algorithm which I solved quickly. Now the project is foolproof, the only possible problem was that of dividing by 0 but that case was handled properly. The only thing that may be called slightly erroneous is that when we perform division and we have an irrational coefficient it will be truncated, but this is normal.

I tested all mathematical operations from the Polynomial class using JUnit Testing. I have 6 methods for testing each having multiple assertEquals() each -the arguments were strings; I used the constructor of the Polynomial class which calls the parsePolynomial() method, so I indirectly tested that one too. The result will be presented below.



```
"C:\Program Files\Java\jdk-20\bin\java.exe" ...
{0=1, 1=3, 2=1}
{0=1, 2=-5, 4=1}
{0=1, 1=3, 2=1}
{0=-1, 1=-3, 2=-1, 4=1}
{0=1, 1=3, 2=1}
```

6. Conclusions

In conclusion, working on the project was great as it provided a valuable insight into the development of a desktop application. I was able to refine my approach towards the entire process of creating an application. The 4 lectures presented by the professor were helpful as I got a better understanding of how to start-by making the use case diagram, and only then start designing classes. I opted for the 1 class approach as it made more sense and I opted to implement the modularity using methods. I tried to keep things as simple as possible, having multiple classes would have overcomplicated the design. The assignment has successfully demonstrated the application of object-oriented design principles, the utilization of graphical user interface (GUI) components, and the handling of various mathematical operations.

What I have learned from making this project:

- **Object-Oriented Design:** I gained a deeper understanding of how to design and structure object-oriented applications, including the creation of classes, methods to encapsulate functionality.
- **Graphical User Interfaces:** I learned about designing user-friendly graphical interfaces using concepts such as menus, input forms, and result displays to enhance user experience.
- **Mathematical Operations:** I explored algorithms and techniques for performing mathematical operations on polynomials, such as addition, subtraction, multiplication, division, derivative, and integration.

Moving forward, there are several areas where the polynomial calculator application could be further developed and improved:

- **Enhanced Functionality**: I consider implementing a root finding method, perhaps even polynomial factorization.
- **Graphical Interface Refinement**: The GUI is far from what is on the market right now, however the interface could be made more user-friendly-one idea is adding buttons for digits to be able to input the polynomial from the GUI not only the keyboard.
- **Performance Optimization:** Optimize algorithms and data structures to improve the performance of mathematical operations, particularly for large polynomials and complex computations.

Overall, I will say that this assignment was a success and played a role in my development as a future software developer.

7.Bibliography

- https://www.uml-diagrams.org/uml-25-diagrams.html
- Programming Techniques lectures, laboratories, and support materials
- https://www.youtube.com/watch?v=vZm0lHciFsQ&ab_channel=CodingwithJohn
- https://www.youtube.com/watch?v=H62Jfv1DJlU&ab_channel=CodingwithJohn
- OOP lectures and laboratory resources from last semester
- https://regexr.com/
- https://www.mathway.com/Algebra
- https://app.diagrams.net/