Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Computer Science Department

Year 2015-2016 Semester II

Programming Techniques

*1st Project*

Student: Dan Toderici

Group: 30422

Contents

[Project specification 3](#_Toc445749074)

[Problem Analysis 3](#_Toc445749075)

[Modeling 3](#_Toc445749076)

[Use Case Diagram 4](#_Toc445749077)

[Design 5](#_Toc445749078)

[Relational Diagram 5](#_Toc445749079)

[Class Structure 7](#_Toc445749080)

[Implementation 7](#_Toc445749081)

[Term Class: public class Term() 7](#_Toc445749082)

[Polynomial Class: public class Polynomial () 8](#_Toc445749083)

[Operations Classes 9](#_Toc445749084)

[Operation class: public abstract class Operation () 9](#_Toc445749085)

[UniOperation class: public abstract class UniOperation() & 9](#_Toc445749086)

[BiOperation class: public abstract class BiOperation() 9](#_Toc445749087)

[Addition class: public class Addition 9](#_Toc445749088)

[Subtraction class: public class Subtraction 9](#_Toc445749089)

[Multiplication class: public class Subtraction 10](#_Toc445749090)

[Division class: public class Division 10](#_Toc445749091)

[Derivation class: public class Derivation 10](#_Toc445749092)

[Integration class: public class Integration 10](#_Toc445749093)

[StringParser class: public class StringParser 11](#_Toc445749094)

[UI class: public class UI 11](#_Toc445749095)

[UIViewController class: public class UIViewController 13](#_Toc445749096)

[Testing and Results 14](#_Toc445749097)

[Conclusion and further developments 14](#_Toc445749098)

[References 15](#_Toc445749099)

# **Project specification**

Propose is to design and implement a system for polynomial processing. Polynomial was specified as having only integer coefficients.

The application, that resulted from design such system, can perform 6 main operations:

1) Addition of two polynomials

2) Subtraction of two polynomials

3) Multiplication of two polynomials

4) Division of two polynomials

5) Derivation of a polynomial

6) Integration of a polynomial

However, all these operations must be executed based on user input. Therefore, a graphical user interface is needed. The UI must provide the user methods to insert the two polynomials and to choose which operations he or she desires to be computed. There will be three fields, one for each polynomial and one for the result, and some buttons, one for each operation.

# Problem Analysis

Finding a way to represent a polynomial in such order to solve our problems will be the first issue. Taking in account that it is an object oriented application, a design that respects this paradigm is required.

Also the operations must be split into groups and therefore extend the same parent class. In addition, not all operations can have as a result a polynomial with integer coefficients, but an method which returns the correct result can be added(ex: integration, division).

# Modeling

In order to keep an object oriented approach of solving this problem, the polynomial must be represented as and ArrayList of terms. Each term has a coefficient and a power. Starting from that, all operations are having as a result a polynomial.

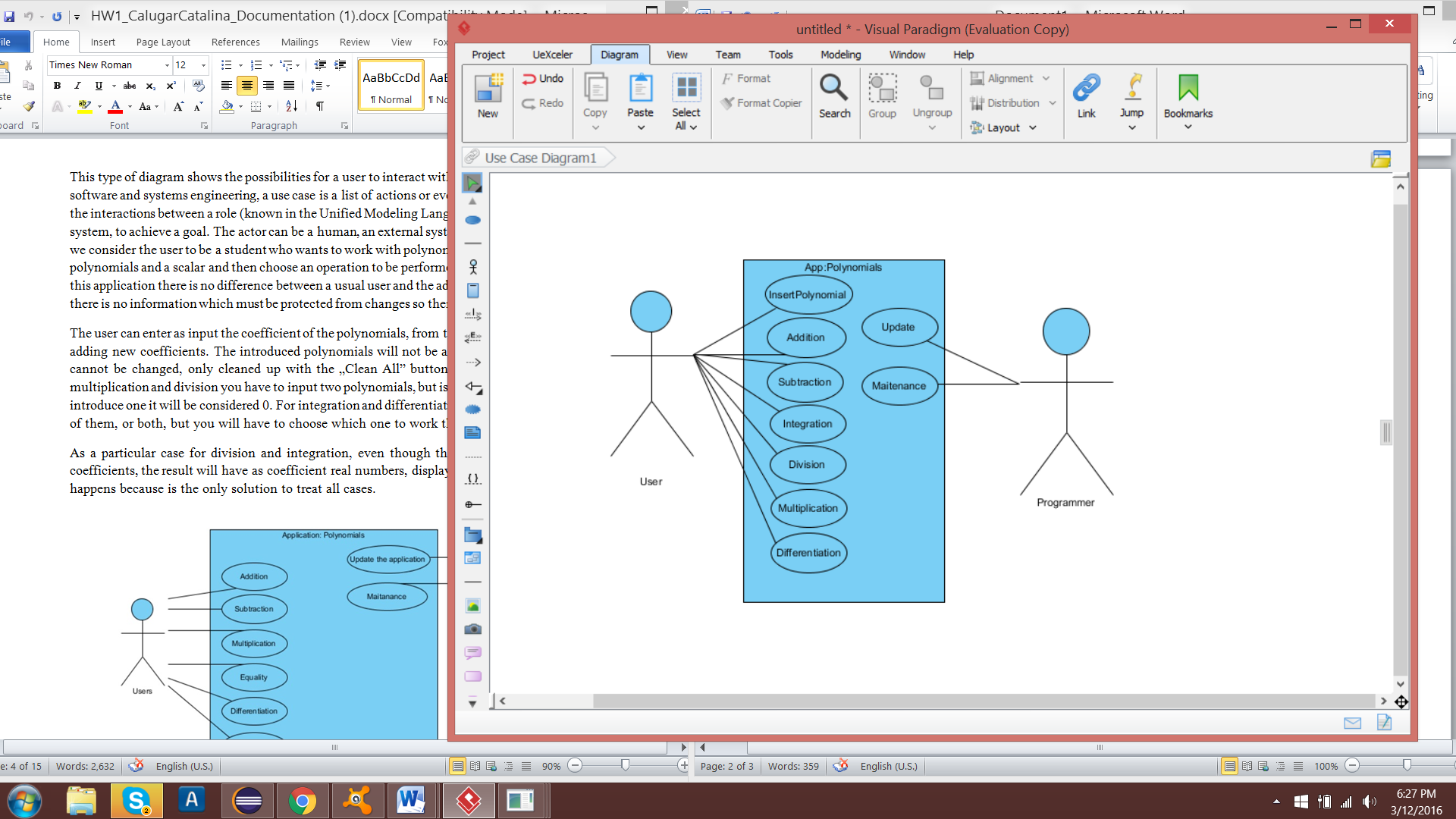
## Use Case Diagram

This type of diagrams have an important role in showing how an user may interact with some features of a certain application. In software and systems engineering, a use case is a list of actions or steps, that define the interaction between a role and a sytem( in the Unified Modeling Language the role is also known as an actor) in order to achieve a goal.

In this case, the actor can be a student who wants to check if he solved his homework on polynomials corectly. He or she inserts the polynomials(in the normal form, ex:2x^2-5x^4) and click on the “Submit” button, than the program parses the into and transforms the String into an array of terms. If one polynomial has to changed, the student must only press the “Cancel” button and a new polynomial can be inputed.

As far as the operations are concerned, the user only as to choose which respective button to click on and the result is immediately outputed in the “Result” textfield.

To avoid division by zero exception, the null polynomial cannot be submitted. It is essential to know that there is no use in dividing or multiplying by 0.



The Use Case Diagram of the application

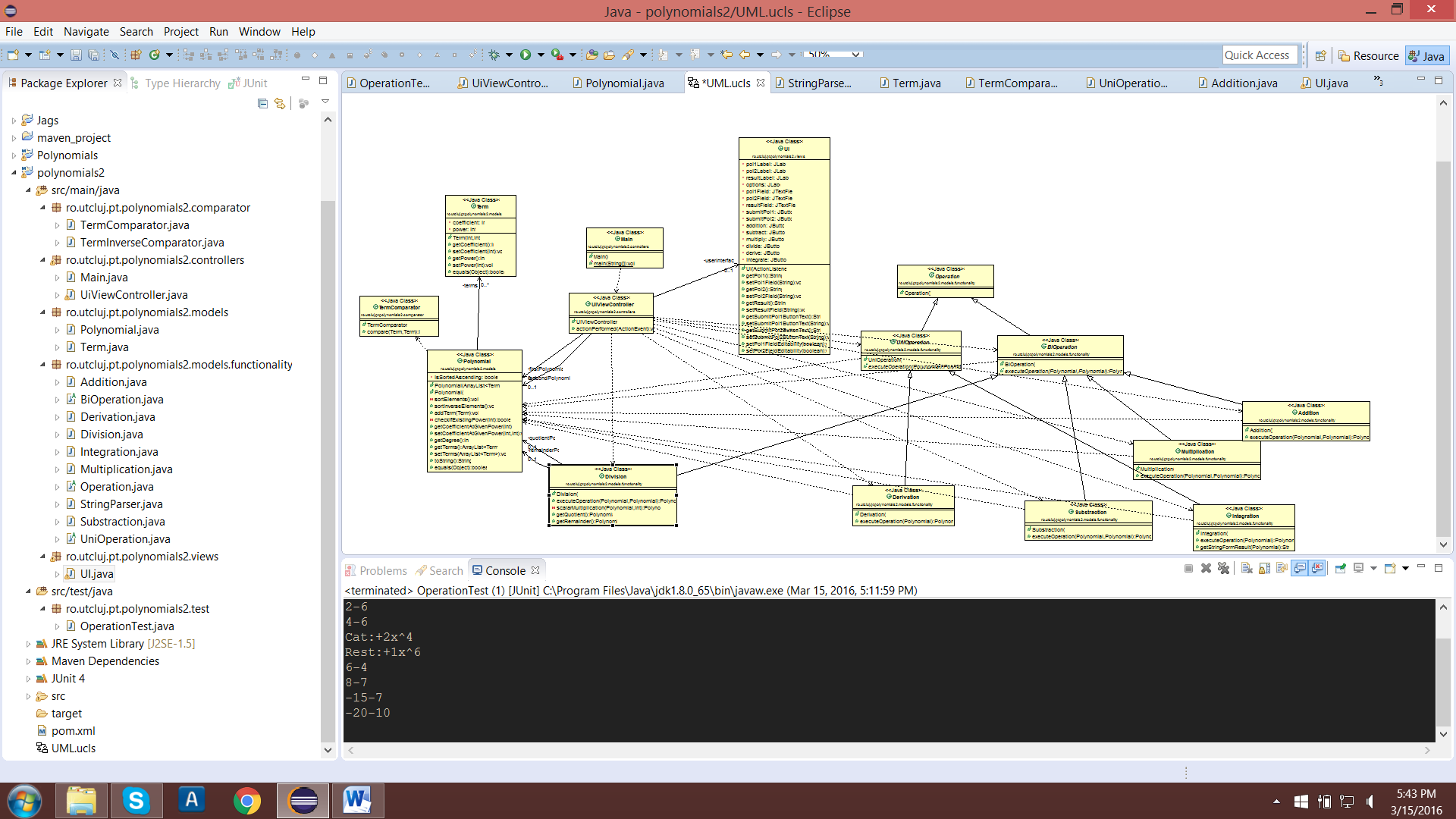
# Design

## Relational Diagram

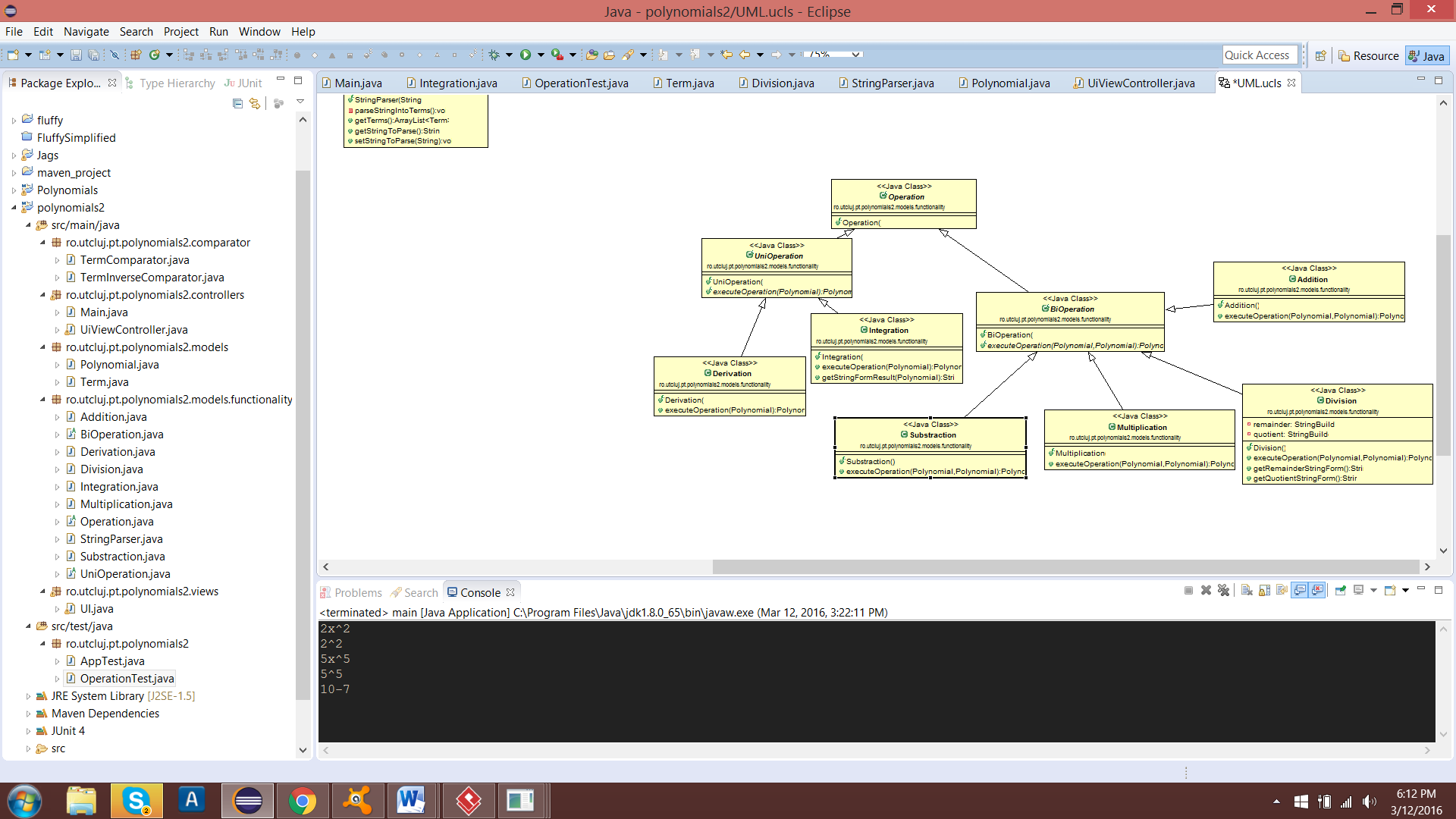
The entity-relational model is a data model that describes aspects of a business domain or its process requirements, in such an abstract way that it can lead to being implemented in a database such as a relational database. The main components of ER models are entities and the relationships that can exist between them.

In most cases the ER design is the first step in solving any OOP problem. I have chosen to implement each operation into a separate class and also to add a class for testing. Also, I have used a utility class “StringParser” to modify the input from the user, in order to generate a polynomial.Not to forget that there is a special type of association between Term and Polynomial, called composition.

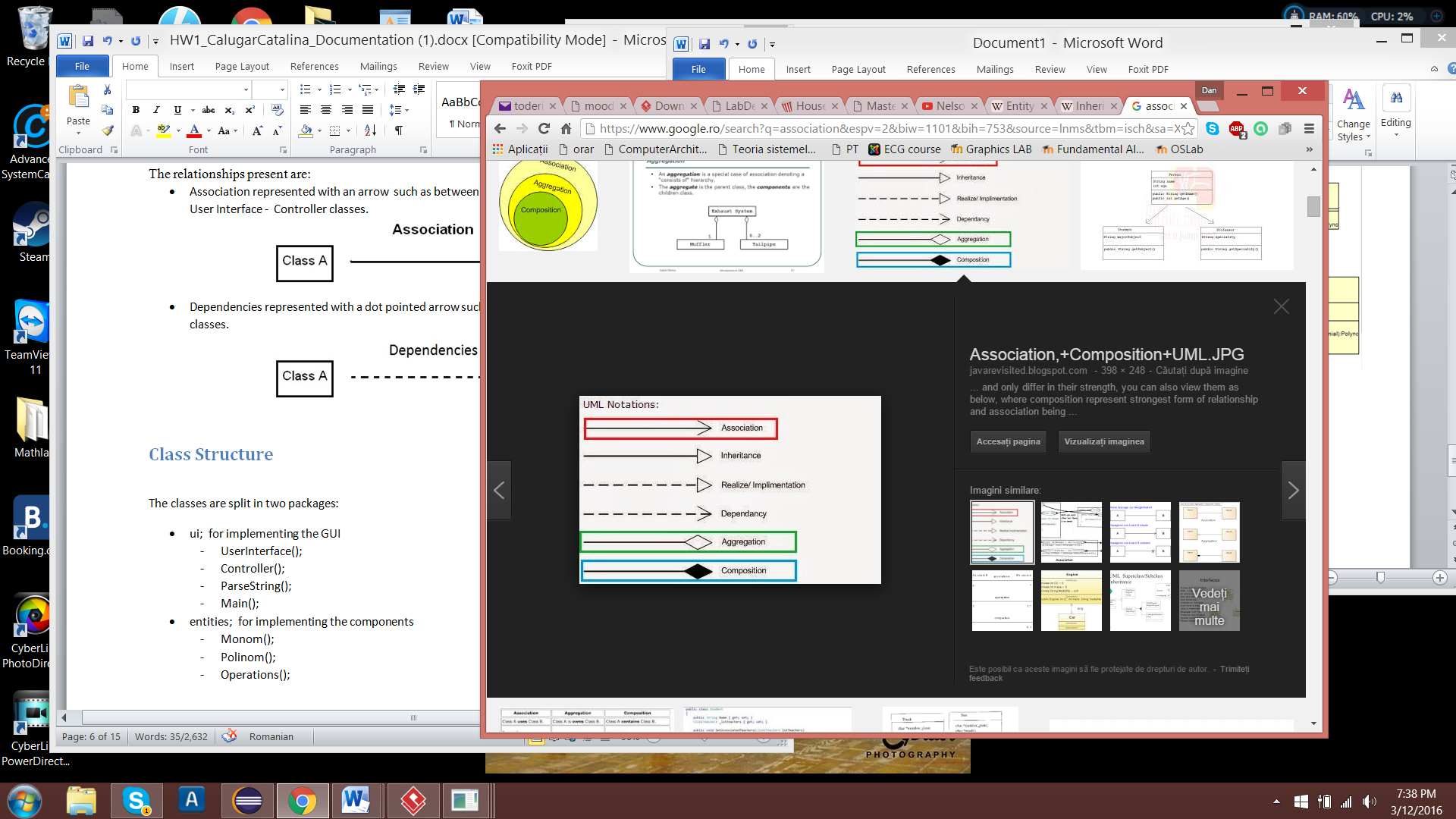
This is the relationship diagram containing associations and dependencies type relations:



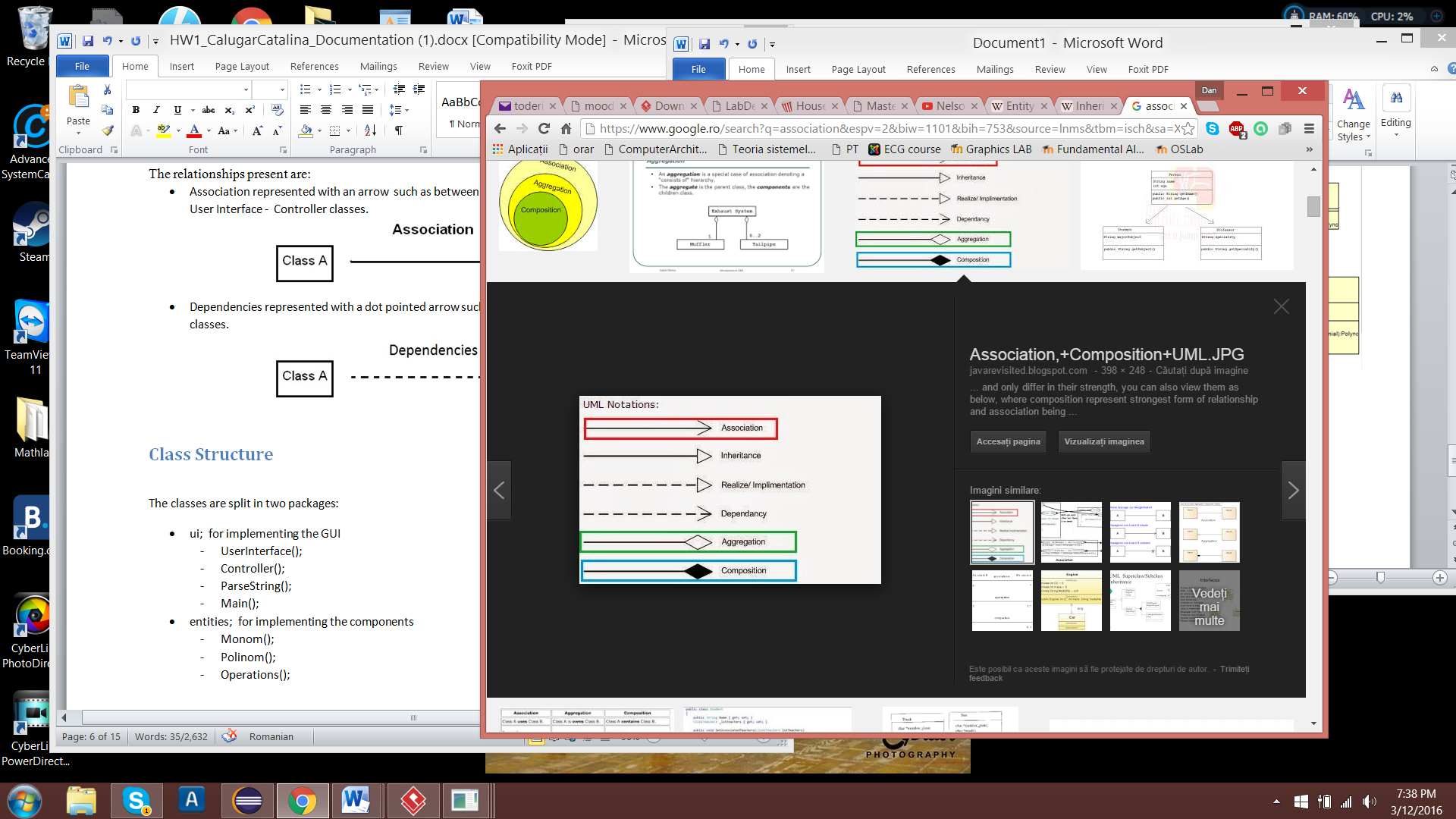
In addition, in this part of the diagram there is presented a special type of association that is called inheritance:

****The relationships that are shown in the diagrams are:

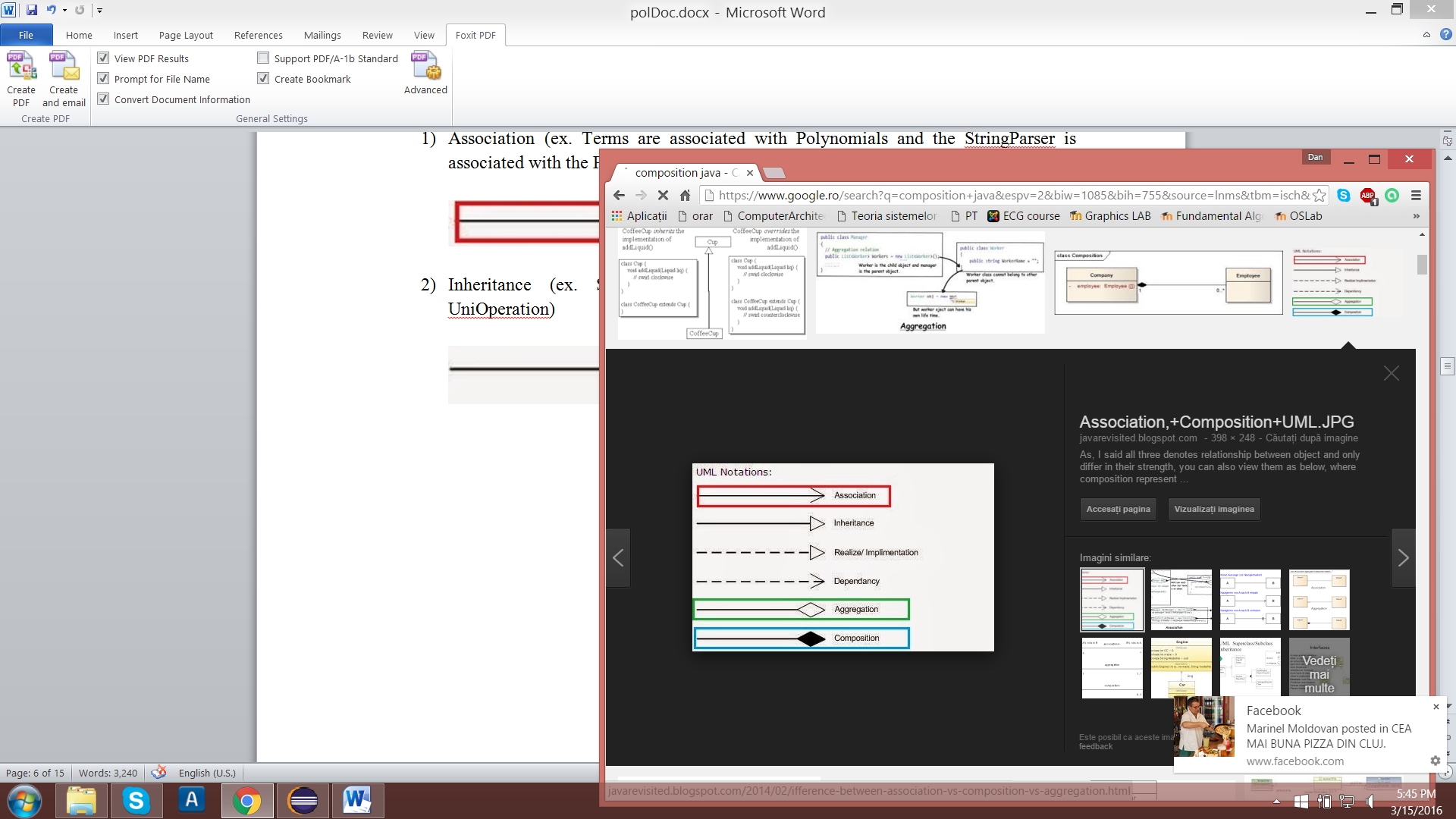
1. Association (ex. Terms are associated with Polynomials and the StringParser is associated with the Polynomial.



1. Inheritance (ex. Subtraction inherits BiOperation and Derivation inherits UniOperation)



1. Composition (ex. Polynomial is composed of Terms)



## Class Structure

The classes are split into numerous packages:

1. ro.utcluj.pt.polynomials2.models:

-Polynomial

-Term

1. ro.utcluj.pt.polynomials2.functionality:

-UniOperation

-BiOperation

-StringParser

-Addition

-Derivation

-Integration

-Multiplication

-Operation

-Subtraction

-Division

1. ro.utcluj.pt.polynomials2.controllers:

-Main

-UiViewController

1. ro.utcluj.pt.polynomials2.view:

-Ui

1. ro.ulcluj.pt.polynomials2.comparator:

-TermComparator

-TermInverseComparator

1. ro.utcluj.pt.polynomials2.test:

-OperationTest

# Implementation

### Term Class: public class Term()

This represents the main entity which will form the ArrayList in class Polynomial() in order to have a good and clear representation polynomial. Because of this, the Term class contains only 2 attributes, the coefficient and the power. The constructor “**public** **Term** (**int** coef, **int** power)“ initializes the instance with the help of 2 set methods.

Due to the principle of encapsulation both properties are private and in order to reach and manipulate the setter and getter methods where used.

The overridden method, “**public** **Boolean** **equals** (Object a)” will be used later on in the Polynomial () class implementation. The manner in which this method was overridden respects the basic principle of comparison and equality, by checking is the parameter is firstly instance of class Term() and only after the cast to Term its properties are checked.

## Polynomial Class: public class Polynomial ()

As afore mentioned, Polynomial class has at its heart an ArrayList of Terms by using generics and the java convention for the generics, after new Type<> nothing must be specified due to the fact that eclipse already knows the type desired:

**private** ArrayList<Term> terms = **new** ArrayList<Term>() ;

The second and only remaining attribute is **private** **boolean** isSortedAscending .This attribute can be used to develop a feature of displaying the polynomial in ascending or descending order for example(5x^6 + 4x^2 instead of 4x^2 + 5x^6).

The constructor **public** **Polynomial** (ArrayList<Term> list) only sets the value because both properties have already allocated memory.

Method setTerms (ArrayList <Term> terms) use another method addTerm(Term term), where existence of that term is check. By existing I mean that the coefficient of that power is different from 0, and if it exists than the coefficient value is the resulting sum of the old coefficient plus the newly added one.

In order to ease our access to specific coefficients and without using a HashSet in the implementation instead of an ArrayList, the **getCofficientAtGivenPower** (int power) was used. This, as easily observed from its name, returns the coefficient at a specific power and if there is no Term having that power, 0 is the answer.

The method **getDegree**() is computed by fetching the power of the last element in the array of terms.

Coming to the point of discussing overridden methods, the **boolean equals**(Object a) , returns whether polynomial is equal to another term-wise. Not to forget about **String toString**(), which is used later on in the UI for displaying the polynomial in its natural representations (ex: -5x^6 + 3x^2)

## Operations Classes

### Operation class: public abstract class Operation ()

Is only a marker classes and has no implementation due to the key word abstract.

### UniOperation class: public abstract class UniOperation() &

### BiOperation class: public abstract class BiOperation()

In order to have a good grouping of the Operation classes, they were split into two groups. UniOperation which, as observed from its only method, receives only one Polynomial object as a parameter and BiOperation , as you might have guessed already, receives 2 objects of type Polynomial as parameters.

### Addition class: public class Addition

The first and the most basic of operations is addition. This class extends BiOperation class and therefore has 2 parameters transmitted to its only overridden method: **public** Polynomial **executeOperation**(Polynomial a, Polynomial b) . This is a must implement method, due to the fact that the class which was inherited is abstract.

The procedure of adding 2 polynomials is quite simple and the result must be a polynomial as well. First step was to initialize the result object with the terms of the first polynomial. Afterwards to iterate the second polynomial, using an enhanced loop (**for** Term **t**: b.getTerms() ), and check if the term having the power of term t is contained already in the result. If not t is added to result, else the coefficient of t is added to the existing term.

### Subtraction class: public class Subtraction

As well known, from a mathematical stand-point, subtraction is the opposite of addition and consequently is implemented as the reverse of it. However, you must keep in mind that the second parameter is subtracted from the first parameter, not the other way around. So the implementation is similar and due to the inheritance of BiOperation class.

### Multiplication class: public class Subtraction

Passing on to a more demanding operation, multiplication needs 2 enhanced loops in order to fetch terms of first polynomial and the second polynomial. The algorithm implies that any term of the first polynomial should be multiplied with any of the terms in the second and result in a new term. The new term has a coefficient equal to the multiplication of the 2 terms’ coefficients and the power equal to the addition of their powers.

### Division class: public class Division

Derivation is the most difficult operation to implement, however keeping in mind that the result should be a polynomial that has integer coefficients, representing the quotient of the division, is not too difficult to compute the remainder. In the process of computing the quotient the divider changes after every step, and a multiplication with scalar is needed. The problem can be solved by introducing a method that receives a polynomial and a scalar value and returns another polynomial (**private** Polynomial **scalarMultiplication** (Polynomial a, **int** scalar).The algorithm is the classic one and has nothing special.

### Derivation class: public class Derivation

Is the first UniOperation and as expected it has only one overridden class (**public** Polynomial **executeOperation**(Polynomial a)). Differentiation if quite simple to implement, once a polynomial is passed on as a parameter, all its terms are fetch in an enhanced loop. The new terms are formed by having the power as a result of decrementing the old power by one and the coefficient equal with multiplication between the term’s old power and its coefficient.

### Integration class: public class Integration

Due to the fact that integration is the opposite of derivation, one must think that the two executeOperation methods are symmetric as in the case of addition and subtraction. However, having the limitation of only using integer coefficients values as 0.76 or even 0.99 are rounded automatically by 0. This problem appears because of the floor operation, which is called upon a double value, when parsing it to int.

Moreover, we can add 1 to the obtained coefficient before converting it to integer, but it will not be an exact representation compared to the real value.In this matter another method was needed: **public** String **getStringFormResult**(Polynomial a) . After computing the coefficient, in order to keep it in double form but not display all the digits a NumberFormat instance is needed. Methods such as setMininumFractionDigits(int nb) and setMaximumFractionDigits( int nb) ease our work. Finally, a string is created by applying the numberformat using objectinstance.format(doubleNb) instruction and the result is than appended to the resulting String, which will be returned at the end of all computations.

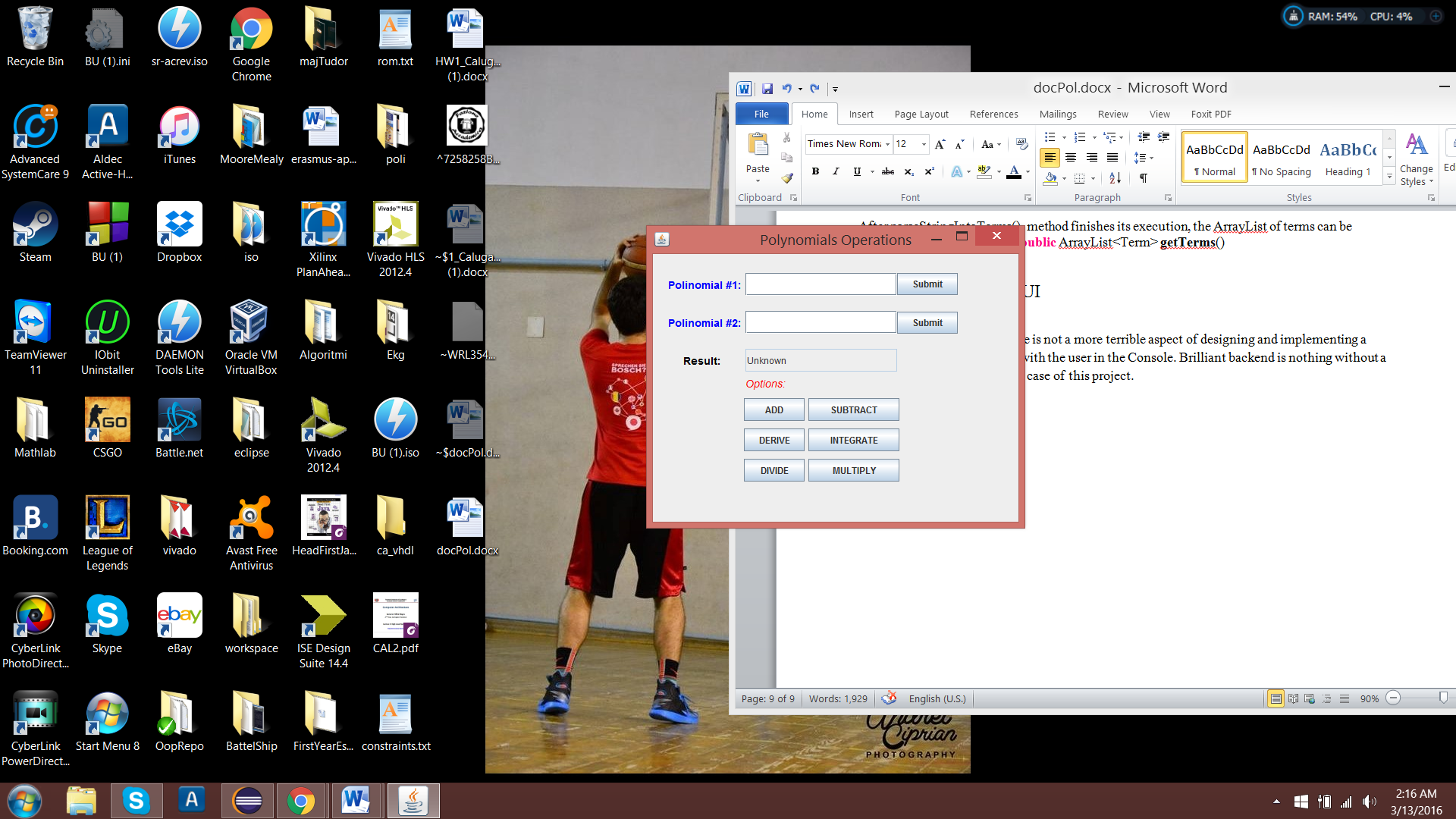
## StringParser class: public class StringParser

Taking in consideration that we want to simulate a real computing environment, we cannot oblige our user to introduce a polynomial only by specifying the power and then input all the coefficients. In order to make him visualize his or her polynomial, the input format must be in a real manner such as in any other Math related applications (ex: 2x^4 - 6x^4). Nevertheless, the input will be a String, so how can one grasp the polynomial, if its only reliable constructor needs an array of terms? The answer is quite simple, create a class that receives the String as a parameter for one of its methods and internally stores the array of terms. I have achieved this by using stringObject.replaceAll(String substring1, String substring2) method and also stringObject.split(Strings substring ).

After parseStringIntoTerms() method finishes its execution, the ArrayList of terms can be retrieved by using the getter: **public** ArrayList<Term> **getTerms**()

## UI class: public class UI

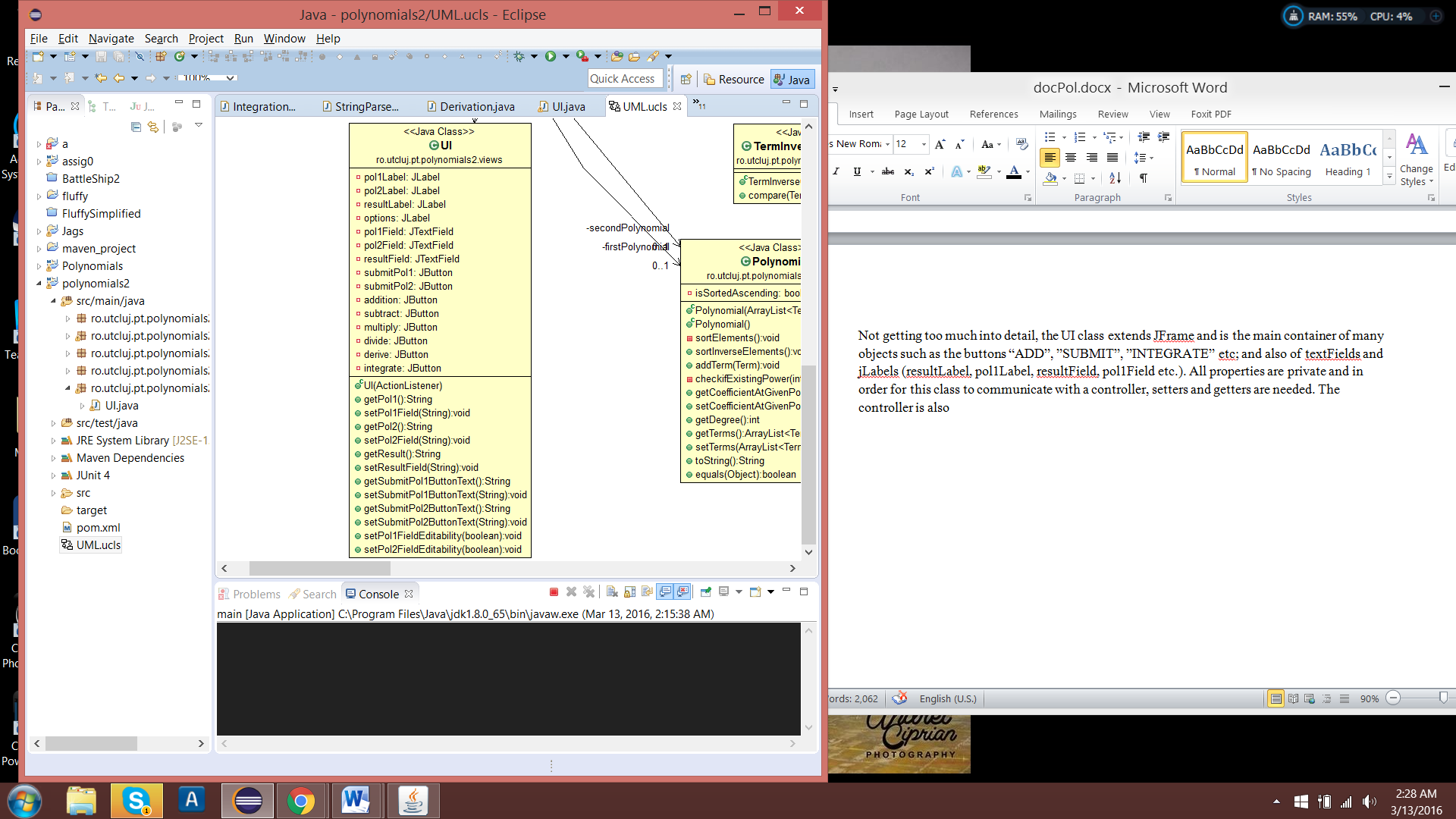
From a visual standpoint, there is not a more terrible aspect of designing and implementing a project which communicates with the user in the Console. Brilliant backend is nothing without a decent frontend and that is the case of this project.



Although it appears simplistic, the UI Class hides one specific aspect in its implementation that cannot be observer only by looking at the above image. The fact that it only displays Strings as the Result of the operations and the 2 polynomials. Having in mind that a class should do only one thing and that task should be performed at its best of capabilities, this class would pass with flying colors.

Not getting too much into detail, the UI class extends JFrame and is the main container of many objects such as the buttons “ADD”, ”SUBMIT”, ”INTEGRATE” etc; and also of textFields and jLabels (resultLabel, pol1Label, resultField, pol1Field etc.). All properties are private and in order for this class to communicate with a controller, setters and getters are needed.

The specific location of each component was set with setBounds method, because I found it more efficient and flexible than using predefined layouts. Also the colors of the foreground and background of some components was changed by using setBackground and setForeground methods and some default instances of Color class such as Color.red, Color.blue etc.



The controller is also its ActionListener, which means that when a button is pushed this component handles the event. Handling an event, means executing a set of instruction if the event occurs.

That brings us to a very known and used design pattern called Model – View – Controller (MVC).This is a software architectural pattern mostly (but not exclusively) for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. The three main components are:

1. The model (in this particular case the polynomials and also the operations classes, although it) manages the data, logic and rules of the application. It captures the behavior of the app in terms of its problem domain, independent of the user interface
2. The view only displays to the user based on changes in the model.
3. The controller accepts input and converts it to commands for the model or view.

## UIViewController class: public class UIViewController

As the name of the sub-chapter mentions, this class is the controller of the UI instance. It implements the ActionListener interface and so in its implementation **public** **void** **actionPerformed**(ActionEvent e) method must appear. This method is called from the UI by all its buttons, when they are pushed. Then the Controller, depending on the buttonSource Text or ActionCommand, executes a set of instructions.

The Controller is the intermediary between the model and the view and it transmits what type of actions should be done by both components.

# Testing and Results

Testing is an important stage in the development of an application. One can choose the path of money testing and find bugs for random input, or can try to think of corner cases and write specific test cases. In Java, the best way to do it is by writing some test cases in a JUnit class.

OperationTest class will test the functionalities of the project. In case some major bugs are found, some test cases could be run to avoid regression.

Assert package provides us a very special tool, which comes in the form of assertEquals(String message , Object object1, Object object2) method. That explains why the equals (Object o) was overridden in class Polynomial.

Testing the operations is a simple task, if you know the resulting values of some operations on one or two polynomials you just compare the real value with the one returned by the executeOperation method. If one test fails, in the JUnit view box it appears failure and the name of the test that failed and consequently the other test appear as passed.

From that point on, if failed test appear, debugging must be done.

In this case there are methods in the OperationTest class that are mark with the annotation “@Test” to mark them as test cases containers. Each operation has its own method (ex: **public** **void** **subtractionTest**(),**public** **void** **multiplyTest**(), etc.). In each method, there is either a Polynomial object result that is instantiated with the expected value and an asserEquals method call that check the equality between the object and the result of the operation. In some cases, checking the resulting String might be an option(ex: integration and division).

# 

# Conclusion and further developments

This project was a great way of revising the APIE concepts : Abstraction, Polymorphism, Inheritance and Encapsulation. Also to work with ArrayList and generics and solve the problem by creating different operation classes. Not to metion the overriding technique which save me a great of time in comparing two object of the same class or converting the internal structure of Polynomial from an array of terms into a String.

One important aspect was the Term class introduction, that changed the way of storing a polynomial in a more efficient and less memory-consuming way.For example, another way to do it was to set the degree of the Polynomial and then create a list having number of terms equals to the degree. This would have been inefficient in cases such as 10x^1000 + x^6.

Polymorphism and inheritance played an important role in designing a solution to this problem. In abstract classes BiOperation and UniOperation executeOperation() method appeared, which was implemented differently by the subclasses.This might have paved the way for a future Factory pattern implementation.

When it comes to UI there is always the possibility of combining the controller with the view or chossing to separate them. Going the MVC way, I have chosen to have the two components seperatly and it help me better understand the underlying pattern.

Some futher improvemets that might be done would be:

1. Notify the user when the input is inccorect and let him insert other characters instead of ’x’
2. Let the user input a file of polynomials instead of introducing them himself and have a larger number of polynomials to be inputed
3. Reuse the results of the operations in futher computations
4. Add more operations such as evaluate and finding root
5. Develop a better user interface
6. Create a class Polynomial that has Numeric coefficients and enable operations on different primitive numeric data type

# References

<https://en.wikipedia.org/wiki/Use_case>

<http://www.jakowicz.com/what-is-apie/>

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

<https://en.wikipedia.org/wiki/Monkey_test>

<https://en.wikipedia.org/wiki/Unit_testing>