*Technical University of Cluj-Napoca  
Faculty of Automation and Computers  
Department of Computer Science  
2nd Semester 2012-2013*

*Programming Techniques*

*Homework 1*

Pomian Aurelian – 30425

Contents

1. Problem specification …………………………………… 3
2. Example ………………………………………………. …. 4
3. Design ………………………………………………… …. 5
   1. Use Case Diagram ……………………………. ….. 5
   2. Class Diagram ……………………………………… 6
   3. User Interface ……………………………………… 7
   4. Class Design ……………………………………….. 9
   5. Sequence Diagram ………………………………... 13
4. Packages and Interfaces .……………………………… 14
5. Using The Application…………………………………… 15
6. Method Explanation……………………………………… 16
7. Conclusions ……………………………………………… 17
8. References ………………………………………………..17
9. ***Problem Specification***

Homework 1

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

The program will be able to perform some operations that can be done on one or two polynomials. Among these operations there are:

* addition, subtraction, multiplication and division of two polynomials
* derivation and integration of a polynomial
* calculating polynomial for a given value
* polynomial representation using a string for display

The program will have a computer-like graphical interface through which the user can enter data (the two polynomials), update the data and he can select the desired operation through the command buttons.

For derivation, evaluation and integration of a polynomial the user will be able to select which polynomial to perform the operation on from 3 radio buttons (A , B , Result ), as for evaluation and integration they will be able to input a value in another field for the point to which the evaluation is performed and for the constant that will be added after the integration.

1. ***Example***

For the following polynomials :

A = 5X ^3 – 3x^2 – 2

B = -x^3 + 4 x^ 2 - x +2

After the data has been entered the user has to update the polynomial using the Update Button.

The result will always be in float (real number) format except when selecting A or B to be displayed in result.

* A = 5X^3-3X^2-2
* B = -X^3+4X^2-X+2
* A + B = 4.0X^3+X^2-X
* A – B = 6.0X^3-7.0X^2+X-4.0
* A \* B = -5.0X^6+23.0X^5-17.0X^4+15.0X^3-14.0X^2+2.0X-4.0
* A / B = -5.0
  + Remain = 17.0X^2-5.0X+8.0

For Derivate, Evaluate, Integrate polynomial A is selected by default

* Derivate = 15X^2-6X
* Evaluate
  + X = 1 = 0
  + X = 2 = 26
  + X = 3 = 106
* Integrate
  + C = -5 = 1.25X^4-X^3-2.0X-5.0

1. ***Design***
   1. *Use Case Diagram*

A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

**A+B**

**A-B**

**A\*B**

**A/B**

**Integrate**

**Evaluate**

**Derivate**

The user is able to perform A+B , A-B, A\*B, A/B operations with polynomials A and B, and also will be able to perform Derivate, Evaluate(of X), Integrate(+ C) of the selected polynomial ( A (default) , B or Result ).

* 1. *Class Diagram*

A **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

**GUI**

*GUI*

- SA **: String**

- SB **: String**

- SR **: String**

- SRes **: String**

- A **: IntegerP;**

- B **: IntegerP;**

- ResultI **: IntegerP;**

- ResultR **: RealP;**

- select **: int;**

- frame **: JFrame;**

- textField **: JTextField;**

- textField\_1 **: JTextField;**

- textField\_2 **: JTextField;**

- textField\_3 **: JTextField;**

- textField\_4 **: JTextField;**

- textField\_5 **: JTextField;**

- textField\_6 **: JTextField;**

- textField\_7 **: JTextField;**

+ main(String[] args)**:void;**

+GUI;

+initialize**:void;**

**Operations**

*helpers*

- ResultR **: RealP;**

- max(a**:int**, b**:int**) **: void;**

+ evaluateI(A**:IntegerP**, x**:int**)**:int;**

+ evaluateR(A**:RealP**, x**:int**)**:int;**

+ addP(A**:RealP,** B**:RealP**)**:RealP;**

+ subP(A**:RealP,** B**:RealP**)**:RealP;**

+ mulP(A**:RealP,** B**:RealP**)**:RealP;**

+ divP(A**:RealP,** B**:RealP**)**:RealP;**

+ integrate(A**:RealP**, C**:int**)**:RealP;**

+ derivateI(A**:IntegerP**)**:IntegerP**;

+ derivateR(A**: RealP**)**: RealP**;

+ toPolynomI(S**:String**):**IntegerP;**

+ toPolynomR(S**:String**): **RealP;**

**-** coefSel(coef**:int**)**:String;**

**-** coefSel(coef**:float**)**:String;**

+ toStringI(A**:IntegerP**)**:String;**

+ toStringR(A**: RealP**)**:String;**

+ toRealP(A**:IntegerP**)**:RealP;**

+ toIntegerP(A:**RealP**)**:IntegerP;**

+getRemain **:RealP;**

+setRemain(remain**: RealP**)**:void;**

**IntegerP**

*main*

+ coef**:int[];**

- initP**:void;**

**RealP**

*main*

+ coef**:float[];**

- initP**:void;**

**Polynom**

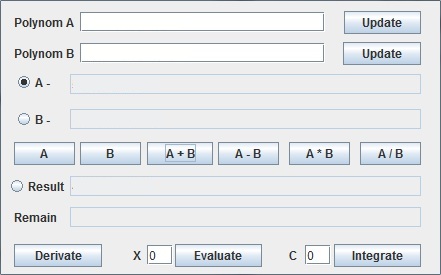
*main*

- n**:int;**

+ setN(n**:int**)**:void;**

+ getN **:int;**

* 1. *User Interface*

**

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

This is the UI ( User Interface ). It has a clean and arranged design.

Contains:

* 4 text fields for input for polynomials A and B , evaluate point X and integrate constant C.
* 4 text field for output for polynomial display A and B, Result and Remain for division
* 2 Update buttons for updating the polynomials A and B.
* 3 Radio buttons for selecting for which polynomial to apply Derivate, Evaluate, Integrate
* 6 Buttons for all 4 basic operations performed on polynomials A and B, also for displaying them
* 3 Buttons for 3 more complex operations on selected polynomial by the radio buttons, Derivate, Evaluate and Integrate.
* 3 Labels for the 2 polynomials and for remain field input texts “Polynom A” , “Polynom B” and “Remain”.

**The polynomial input format:**

* Element format: (sign)[coefficient]’X^’[grade]
* If grade is 0 or 1 it can be absent
* If coefficient is 0 or 1 it can absent
* If sign is positive it can be absent
* For grades 0 and 1 ‘^’ character can be absent
* ‘X’ is not required to be uppercase
* Spacing doesn’t matter

Example:

* Good : -5x ^ 3 – 3X^2+ 2x – 2 or +1x^2+2X^1 -2x^0
* Bad : -5X3 - 3x^2 +2x1 -2^0

For a more in-depth explanation on StringToPolynomial

see **6.a. toPolynom …………. 15**

* 1. *Class Design*
* **GUI**

Package: GUI

Purpose of this class is to create and display a graphical user interface in order to an easier way to use the application.

Extends class **Operations**.

The function **public** **static** **void** main(String[] args) is in this class which makes the whole application running.

**public** **static** **void** main(String[] args) {

GUI window = **new** GUI();

window.frame.setVisible(**true**);

}

As a result of this run it will create a new instance of class GUI (window) which will be initialized by the constructor and will set the frame to be visible which will make the window visible and ready to use.

This class imports the following libraries: swing and awt.

**Swing** is a GUI widget toolkit for Java an API for providing a graphical user interface (GUI) for Java programs.

The **Abstract Window Toolkit** (**AWT**) is Java’s original platform-dependent windowing, graphics, and user-interface widget preceding Swing.

Contains a private JFrame (frame) which represents the main window of the user interface and contains all components (labels, buttons, text fields).

Constructor: **public** GUI() {

initialize();

}

The initialize() method adds all components to the frame and creates Action Listener (what to do when a button is clicked) for each button.

Contains 25 components.

see **3. c. User Interface …………… 7**

Code example for the Derivate button inside the initialize() method:

btnDerivate.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent arg0) {

**if** (select == 0) {

ResultI = *derivateI*(A);

SR = *toStringI*(ResultI);

textField\_3.setText(SR);

textField\_6.setText("");

} **else** **if** (select == 1) {

ResultI = *derivateI*(B);

SR = *toStringI*(ResultI);

textField\_3.setText(SR);

textField\_6.setText("");

} **else** {

SRes = textField\_3.getText();

ResultR = *toPolynomR*(SRes);

ResultR = *derivateR*(ResultR);

SRes = *toStringR*(ResultR);

textField\_3.setText(SRes);

textField\_6.setText("");

}

}

});

* select is used to determine which polynomial field is select (A , B or Result).
* textField\_3 is the Result text field where the latest result will be displayed.
* textField\_6 is the Remain text field which will be reset to empty text after each operation in order maintain a clean look of the interface in case the division operation was previously used.
* **Operations**

Package: helpers

This class contains all the methods we are using with, on and to polynomials.

The purpose of this class is to give the code a more organized structure and easier to read.

**Methods:**

* **addP(A, B)** – adds polynomial A with polynomial B
* **subP(A, B)** – subtracts polynomial B from polynomial A
* **mulP(A, B)** – polynomial A times polynomial B
* **divP(A, B)** – divides polynomial A to polynomial B
* **toStringI(A)/toStringR(A)** – converts polynomial A (int or real) to string representation
* **toPolynomI(S)/toPolynomR(S)** – converts string S to polynomial representation format that I’m using (int or real)
* **evaluateI(A,x)/evaluateR(A,x)** – evaluates the polynomial A(int or real) in a given point x.
* **derivateI(A)/derivateR(A)** – derivates polynomial A(int or real)
* **integrate(A,C)** – integrates polynomial A with a given constant C

The following methods have a more in-depth explanation: **toPolynom**.

see .**6.a. toPolynom …………. 15**

Additional helping methods:

* **coefSel(coef)** – used for representing the coefficient along with the ‘X’ in string according to their sign
* **max(a,b)** – used for returning the max value of the two integers (a and b)
* **getRemain()/setRemain(remain)** – used when divP operation is used. remain is a polynomial with real coefficients and stores the remainder of the division operation.

This class contains static methods and is extended by **GUI** in order to use them.

* **Polynom**

Package: main

This class contains a single private attribute **n,** and it represents the order(grad) of the polynomial.

Contains 2 public methods for getting and setting the n:

* **getN()**
* **setN(n)**

Is extended by classes **RealP** and **IntegerP**.

* **IntegerP**

Package: main

This class contains an array(coef [ ]) with **int** type elements, and represents the coefficients of the polynomial.

**Constructor:**

**public** IntegerP() {

initP();

setN(0);

}

The method initP() is used to initialize all the coefficients with 0.

* **RealP**

Package: main

This class contains an array(coef [ ]) with **float** type elements, and represents the coefficients of the polynomial.

**Constructor:**

**public** RealP() {

initP();

setN(0);

}

The method initP() is used to initialize all the coefficients with 0.0f.

* 1. *Sequence Diagram*

**GUI**

**Operations**

**RealP**

new RealP()

new RealP()

toRealP

A

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

. . . . . . . . . . . . . . . . . .

toRealP

B

addP

new RealP()

ResultR

toStringR

textField\_3

This sequence diagram shows the sequence of methods the program calls when the add button is used.

1. ***Packages and Interfaces***

A Java package is a mechanism for organizing Java [classes](http://en.wikipedia.org/wiki/Class_%28computer_science%29) into [namespaces](http://en.wikipedia.org/wiki/Namespace_%28computer_science%29). Java packages can be stored in compressed files called [JAR files](http://en.wikipedia.org/wiki/JAR_file), allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. A package provides a unique namespace for the types it contains. Classes in the same package can access each other's package-access members.

A package allows a developer to group classes (and interfaces) together. These classes will all be related in some way – they might all have to do with a specific application or perform a specific set of tasks.

For this application the following packages are imported, each of them having a certain role for the proper working of the application. We import them in the Gui Class (most of them relate to the user interface properties):

* **import java.awt:** Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components.
* **import javax.swing:** Typical Swing applications do processing in response to an event generated from a user gesture. For example, clicking on a JButton notifies all ActionListeners added to the JButton. That’s why we use this package for creating the user interface Gui.

1. ***Using the Application***

In order to use the application open Polynom.JAR. This will open a window which generates the GUI class. Thus the user can enter the desired values and selecting the operations by pressing one of the 11 buttons.

Rules for the input data.

see **3. c. User Interface** **(The polynomial input format) ………… 8**

Limits:

* Maximum grade (n) of the polynomial is 30.

Before executing any operations the polynomial data must updated using the Update buttons , otherwise the polynomials will be considered as a grade 0 polynomial with the constant 0.

For the 4 basic operations (add, sub, mul, div) or to display in the Result field the polynomial A or B use the row with the 6 buttons (“A”, “B”, …., “A / B”).

To use the Derivate, Evaluate and Integrate operations, selecting which polynomial to use the operations on is required( A is selected by default), you can choose between A , B and Result polynomials.

Evaluate and Integrate operations require an additional input field for the point to evaluate and constant to add to integrate, if no input is given, 0 is given by default.

1. ***Method explanation***
   1. *toPolynom*

This method is used to convert String to polynomial format.

Return type **RealP**.

Parameters: String S

**public** **static** RealP toPolynomR(String S)

First we convert the string into a more clean and easy to use format using the following :

S = S.replace(" ", "");

S = S.replace("x", "X");

S = S.replace("X^", "X");

S = S.replace("-X", "-1X");

S = S.replace("+X", "+1X");

S = S.replace("-", "+-");

S = S.replace("X+", "X1+");

StringTokenizer Ss = **new** StringTokenizer(S, "+");

This will get rid of the unnecessary spaces and of the character ‘^’, will convert all lower case x to upper case, will add before all ‘-‘ a ‘+’ character in order to properly divide the string into elements.

StringTokenizer is a method from java.util. (java.util.StringTokenizer is imported) that will divide the string into multiple stings by a character given(‘+’ in this case).

Final format of an element: \*\*sign \*[coef] \*‘X’ \*[degree]

\*optional

\*\*depended on coef or X

Notice how an element can be missing all of the format elements, in this case nothing is updated to the polynomial.

At least one of the fields is required to update the polynomial.

This element is further *tokenized* into each field

Ex:

* sign + coef
* X
* Degree

I used *Integer.parseInt()* and *Float.parseFloat()* to convert string to int or float.

1. ***Conclusions***

Achieving such a program may be hard both in terms of algorithms, graphical structure. Although some instructions were easy for implementation, like the addition, subtraction or multiplication, division operation was difficult and some problems in implementation occurred.

Moreover, to read and display a legible polynomial took a special function to transform the data type which was represented by a polynomial into a string of characters to be displayed or read.

For a better performance there should be implemented an auto-update function therefore removing the Update button thus making the application better and easier to use.

1. ***References***

[*http://stackoverflow.com/*](http://stackoverflow.com/)

<http://www.google.ro/>