*Technical University of Cluj-Napoca  
Faculty of Automation and Computers  
Department of Computer  
2nd Semester 2016*

*Programming Techniques*

*Homework 4*

*Student: Ghiurutan Alexandru   
 Group: 30425*

***Contents:***

1. *Problem Specification ................................................................ 3*
2. *Example of working .............................................................. 4*
3. *Design*

*3.1 Use case Diagram …............................................... 5*

*3.2 UML Diagrams …………………………………… 6*

*3.3 Sequence Diagram………………………………….. 7*

*3.4 Package Diagram…………………………………... 8*

*3.5 Classes Design ............................................................ 9*

*3.5.1 Gui Class*

*3.5.2 GenerateReport Class*

*3.5.3 AdminView Class*

*3.5.4 CustomerView Class*

*3.6 Packages and Interfaces............................................. 15*

*3.7**Runnable Interface …............................................... 16*

*3.8 User Interface........................................................... 17*

1. *Using and testing the application …………………………….. 19*
2. *Conclusions …………………..……………………………..... 19*

***1. Problem Specification***

TP Lab–Homework 4

Objective

Design by Contract Programming Techniques

Description

Consider the system of classes in the class diagram below.

1. Define the interface BankProc (add/remove persons, add/remove holder associated accounts,

read/write accounts data, report generators, etc). Specify the pre and post conditions for the

interface methods.

2. Define and implement the classes Person, Account, SavingAccount and SpendingAccount. Other

classes may be added as needed (give reasons for the new added classes).

3. An Observer DP will be defined and implemented. It will notify the account main holder about

any account related operation.

4. Implement the class Bank using a predefined collection which uses a hashtable. The hashtable

key will be generated based on the account main holder (ro. titularul contului). A person may act

as main holder for many accounts. Use JTable to display Bank related information.

4.1 Define a method of type “well formed” for the class Bank.

4.2 Implement the class using Design by Contract method (involving pre, post conditions,

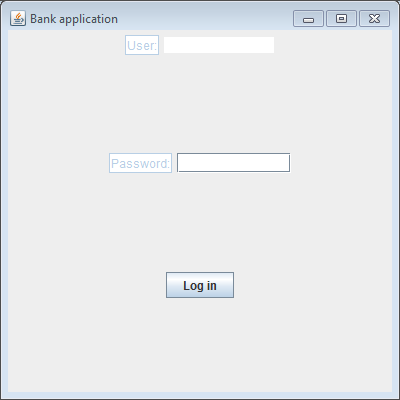
invariants, and assertions).

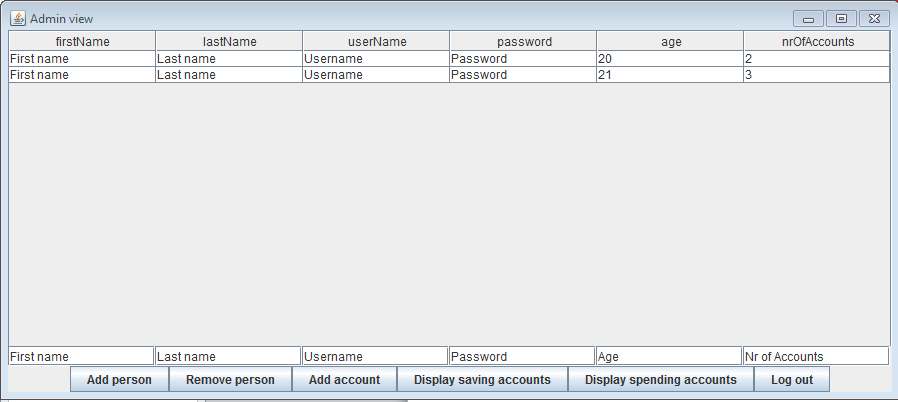
5. Implement a test driver for the system.

6. The account data for populating the Bank object will be loaded/saved from/to a file.

***2. Example of working***

In the picture below ,we can see the logging window from the start , were both the administrator and the customers from the bank can enter .



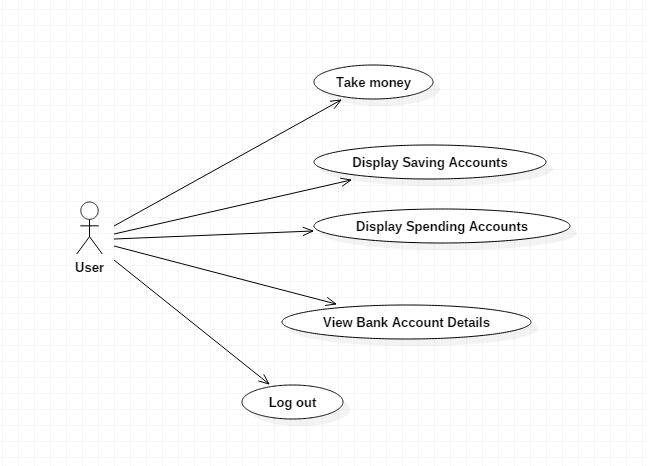
If we log in as admin we can see the window below ,with all the buttons as options like : displaySpendingAccounts , addPerson , removePerson etc . 

***3. Design***

***3.1 Use case Diagram***

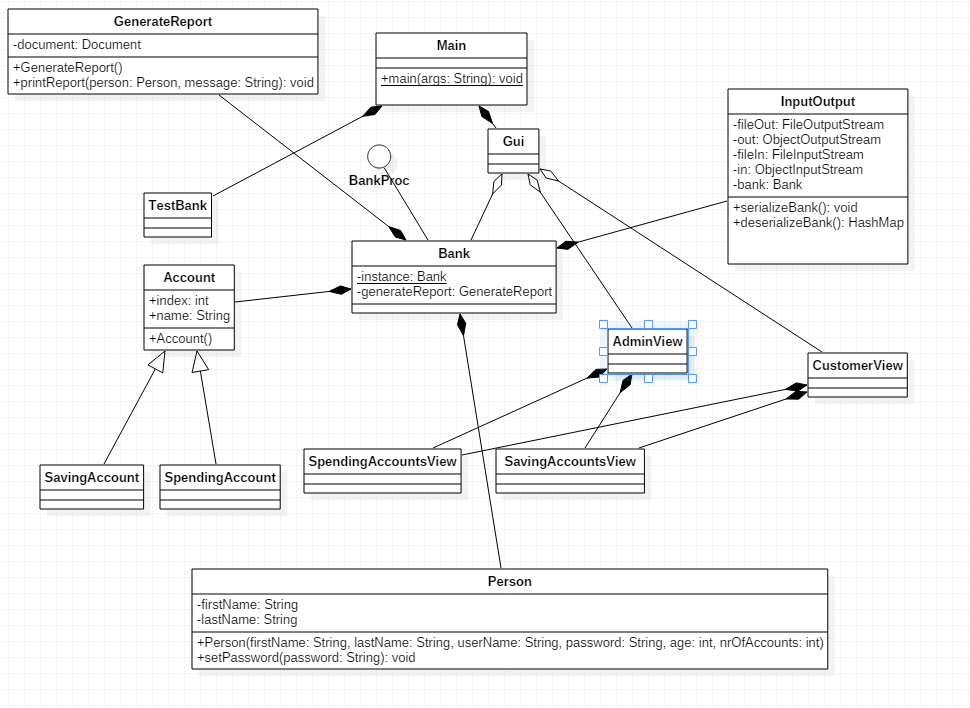
For resolving the problem specifications, I’ve chosen to use four different classes so that the design of the application would be as good as possible. The names of the classes and relations between them are represented in the figure below.

About their use and the reasons why I have chosen to organize the objects as it is written are explained in section 3.2: “Classes Design”



To get a better view related to the attributes of each class, there are below the UML diagrams for each class. Thus, we can see every class with objects and their methods.

***3.2 UML Diagram***



***3.3 Sequence diagram***

SpendingAccount

AdminView

Bank

Create SpendingAccount

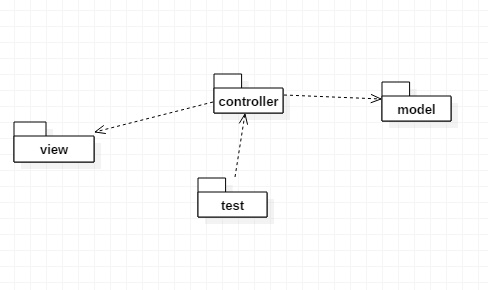
Retrives SpendingAccount

Add holder SpendingAccount

Returns the array of SpendingAccounts

Prints table

***3.4 Package Diagram***

******

***3.5 Classes Design***

1. Gui Class : public class Gui extends JFrame implements ActionListener

This class is designed to create a graphical user interface so the application would be easier to use. We consider the Gui class as a subclass of the predefined class "JFrame" so we can use objects of type "button", "frame" or "panel".

In this class there is declared the function public static void main (String [] args) which makes the whole application running. As a result of this run, on the screen it will open a window through which the user can enter data and perform operations that are possible with this application.

Observe the six TextFields for reading the necessary information at the top of the window. The user can insert any integer, each of them representing:

Number of existing queues in the shop

Service time interval: for each client, the program randomly generates a value in the specific range (contains the minimum and the maximum); this has the significance of time which takes for the customer to be served

Range of arrival: like the “service interval” this is randomly generated for each customer when he/she arrives at a queue.

Simulation time

There are two buttons: one to start the program after input data were introduced (Start) and the other for closing the application (Quit).

Below them there is a space for placing the graphical representation of the queues and the clients for this shop simulation.

The Attributes of the Gui Class:

To achieve the desired GUI we need several types of attributes:

The Constructor of the Gui Class: public Gui ()

It initializes all the attributes declared above; the window with the following elements will be constructed: frame, buttons, text fields. Here there are the frame settings, such as the size, the visibility, title and some predefined operation such as:

public Gui ( ) {

this.setTitle ("Bank application") ;

initializeFields () ;

this.setLayout (new BoxLayout ( getContentPane () , BoxLayout.Y\_AXIS ) ) ;

bank = Bank.getInstance ( ) ;

this.setSize ( 400, 400 ) ;

this.setDefaultCloseOperation ( JFrame.EXIT\_ON\_CLOSE ) ;

this.setVisible ( true ) ;

}

What is more, for every button (operation) there will be added an “ActionListener” which contains the instructions that are needed to be executed in the moment a certain operation button is clicked. For each button there is a different class declared inside the Gui class which implements the “ActionListener” interface.

More information about the “ActionListener” implementation are in the chapter about the user interface where there are presented all the swing components used in the Gui constructor.

The Methods of the Gui Class:

Gui contains methods for achieving operations when you press one of the two existing buttons in the interface. Moreover, it contains the main function which is run every time the application is opened.

initializeFields ( ) method:

This method aims to achieve the interface, this means that within this function we declare a new object's constructor called Gui to achieve early window:

Methods for the button:

* + The Login button:

It declares a class which implements the predefined class called ActionListener. When the Login button is pressed by the user, it generates the the specific message for our option on the userName and password.This is done by constructing a new object of type AdminView or CustomerView. Parameters like number of queues or time intervals from the frame are transmitted to the Shop class constructor.

If the user wants to run the application for other inputs, he needs to enter them in the text fields and press again the Login button. To get the information from the text field we use a function :

String userText = userArea.getText ( ) ;

String passwordText = String.copyValueOf ( passwordArea.getPassword ( ) ) ;

userText.equals(ADMIN\_USERNAME)

passwordText.equals(ADMIN\_PASSWORD)

1. GenerateReport Class: public class GenerateReport

This class is used to generate a pdf document with messages corresponding to the transaction / creating / deleting / modifying the account or the information from a person’s account.

The method that is used in this class is :

public void printReport(Person person, String message) {

try {document = new Document();

PdfWriter.getInstance (document, new FileOutputStream(person.getFirstName() + " " + person.getLastName() + ".pdf"));

document.open();

document.add(new Paragraph(person.toString() + " " + message + "\n"));

document.close();

} catch (DocumentException e) {

e.printStackTrace();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

The Attribute of the GenerateReport Class:

The document corresponding to the user :

private Document document ;

1. AdminView Class: public class AdminView extends JFrame implements ActionListener

AdminView contains methods for achieving operations when you press one of the existing buttons in the interface.

The Attributes of the AdminView Class :

private static final long serialVersionUID = 3774434228678895653L;

private JButton addPerson , removePerson , addAccount , displaySavingAccounts , displaySpendingAccounts , logout ;

private JTable table ;

private DefaultTableModel tableModel ;

private JScrollPane scrollPane ;

private JPanel operations , fields , south ;

private JTextField firstNameField , lastNameField , usernameField , passwordField , ageField , nrOfAccountsField ;

private Bank bank ;

private Object[][] rows ;

private String[] columns ;

private String firstName , lastName , username , password ;

private int age , nrOfAccounts ;

private static String ERROR = " Error " ;

private JTable jT ;

The Constructor of the AdminView Class:

public AdminView() {

this.setTitle("Admin view");

scrollPane = new JScrollPane();

bank = Bank.getInstance();

jT = new JTable();

operations = new JPanel();

operations.setLayout(new BoxLayout(operations, BoxLayout.X\_AXIS));

addPerson = new JButton("Add person");

addPerson.addActionListener(this);

removePerson = new JButton("Remove person");

removePerson.addActionListener(this);

addAccount = new JButton("Add account");

addAccount.addActionListener(this);

displaySavingAccounts = new JButton("Display saving accounts");

displaySavingAccounts.addActionListener(this);

displaySpendingAccounts = new JButton("Display spending accounts");

displaySpendingAccounts.addActionListener(this);

logOut = new JButton("Log out");

logOut.addActionListener(this);

firstNameField = new JTextField("First name", 10);

lastNameField = new JTextField("Last name", 10);

usernameField = new JTextField("Username", 10);

passwordField = new JTextField("Password", 10);

ageField = new JTextField("Age", 10);

nrOfAccountsField = new JTextField("Nr of Accounts", 10);

fields = new JPanel();

fields.setLayout(new BoxLayout(fields, BoxLayout.X\_AXIS));

fields.add(firstNameField);

fields.add(lastNameField);

fields.add(usernameField);

fields.add(passwordField);

fields.add(ageField);

fields.add(nrOfAccountsField);

operations.add(addPerson);

operations.add(removePerson);

operations.add(addAccount);

operations.add(displaySavingAccounts);

operations.add(displaySpendingAccounts);

operations.add(logOut);

south = new JPanel();

south.setLayout(new BoxLayout(south, BoxLayout.Y\_AXIS));

south.add(fields);

south.add(operations);

this.add(south, BorderLayout.SOUTH);

initializeNorthSection();

this.setSize(900, 400);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setVisible(true);

}

The methods of the AdminView Class:

private void initializeNorthSection() {

table = createGeneralTable(bank.getAllPersons());

scrollPane = new JScrollPane(table);

this.add(scrollPane, BorderLayout.NORTH);

}

private JTable createGeneralTable(Collection<Object> data) {

if (data == null) {

return null;

}

int listLength;

Object[] list = data.toArray();

if (list.length != 0) {

Field[] fields = list[0].getClass().getDeclaredFields();

Field.setAccessible(fields, true);

rows = new Object[list.length][fields.length - 1];

Object[] row = new Object[fields.length - 1];

columns = new String[fields.length - 1];

int i = 0;

for (Field field : fields) {

if (i == columns.length) {

break;

}

columns[i++] = field.getName();

}

tableModel = new DefaultTableModel(rows, columns);

tableModel.setRowCount(0);

jT.setModel(tableModel);

jT.setEnabled(false);

listLength = list.length;

for (i = 0; i < listLength; i++) {

int j = 0;

for (Field field : fields) {

if (j == row.length) {

break;

}

try {

Object fieldValue = field.get(list[i]);

row[j++] = fieldValue;

} catch (IllegalArgumentException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

}

}

tableModel.addRow(row);

}

}

return jT;

}

private void updateTable() {

table = createGeneralTable(bank.getAllPersons());

}

private boolean checkFields() {

if (firstNameField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the first name of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else if (lastNameField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the last name of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else if (usernameField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the username of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else if (passwordField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the password of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else if (ageField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the age of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else if (nrOfAccountsField.getText().equals("")) {

JOptionPane.showMessageDialog(this, "You didn't enter the nr of accounts of a person.", ERROR,

JOptionPane.ERROR\_MESSAGE);

return false;

} else {

return true;

}

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == addPerson) {

if (checkFields()) {

firstName = firstNameField.getText();

lastName = lastNameField.getText();

userName = usernameField.getText();

password = passwordField.getText();

age = Integer.valueOf(ageField.getText());

nrOfAccounts = Integer.valueOf(nrOfAccountsField.getText());

bank.addPerson(new Person(firstName, lastName, userName, password, age, nrOfAccounts));

updateTable();

}

} else if (e.getSource() == removePerson) {

if (checkFields()) {

firstName = firstNameField.getText();

lastName = lastNameField.getText();

userName = usernameField.getText();

password = passwordField.getText();

age = Integer.valueOf(ageField.getText());

nrOfAccounts = Integer.valueOf(nrOfAccountsField.getText());

bank.removePerson(new Person(firstName, lastName, userName, password, age, nrOfAccounts));

updateTable();

}

} else if (e.getSource() == addAccount) {

if (checkFields()) {

firstName = firstNameField.getText();

lastName = lastNameField.getText();

userName = usernameField.getText();

password = passwordField.getText();

age = Integer.valueOf(ageField.getText());

nrOfAccounts = Integer.valueOf(nrOfAccountsField.getText());

Person person = new Person(firstName, lastName, userName, password, age, nrOfAccounts);

if (bank.containsPerson(person)) {

addAccount(person);

updateTable();

}

}

} else if (e.getSource() == displaySavingAccounts) {

if (checkFields()) {

firstName = firstNameField.getText();

lastName = lastNameField.getText();

userName = usernameField.getText();

password = passwordField.getText();

age = Integer.valueOf(ageField.getText());

nrOfAccounts = Integer.valueOf(nrOfAccountsField.getText());

Person person = new Person(firstName, lastName, userName, password, age, nrOfAccounts);

new SavingAccountsView(person, bank.getPersonSavingAccounts(person));

updateTable();

}

} else if (e.getSource() == displaySpendingAccounts) {

if (checkFields()) {

firstName = firstNameField.getText();

lastName = lastNameField.getText();

userName = usernameField.getText();

password = passwordField.getText();

age = Integer.valueOf(ageField.getText());

nrOfAccounts = Integer.valueOf(nrOfAccountsField.getText());

Person person = new Person(firstName, lastName, userName, password, age, nrOfAccounts);

new SpendingAccountsView(person, bank.getPersonSpendingAccounts(person));

updateTable();

}

} else if (e.getSource() == logOut) {

this.dispose();

new Gui();

}

}

1. CustomerView Class: public class CustomerView extends ActionListener

It is similar to the AdminView class with some constraints and less operations.

The Attributes of the CustomerView Class:

private JButton takeMoney, displaySavingAccounts, displaySpendingAccounts, logOut, viewBankAccountDetails;

private Bank bank;

private JPanel operations;

private Person person;

private String[] selectionValues = { " Spending account " , " Saving account " } ;

private String initialSelection = "Spending account";

The Constructor of the CustomerView Class:

Initialize a customer's attributes ( is called from the Gui class when we log in with a new userName and password ) .

public CustomerView(Person person) {

this.setTitle("Customer view");

this.setSize(900, 400);

bank = Bank.getInstance();

this.person = person;

operations = new JPanel();

operations.setLayout(new BoxLayout(operations, BoxLayout.X\_AXIS));

takeMoney = new JButton("Take money");

takeMoney.addActionListener(this);

displaySavingAccounts = new JButton("Display saving accounts");

displaySavingAccounts.addActionListener(this);

displaySpendingAccounts = new JButton("Display spending accounts");

displaySpendingAccounts.addActionListener(this);

viewBankAccountDetails = new JButton("View Bank account details");

viewBankAccountDetails.addActionListener(this);

logOut = new JButton("Log out");

logOut.addActionListener(this);

operations.add(takeMoney);

operations.add(displaySpendingAccounts);

operations.add(displaySavingAccounts);

operations.add(viewBankAccountDetails);

operations.add(logOut);

this.add(operations, BorderLayout.SOUTH);

this.setVisible(true);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

The methods of the CustomerView Class:

public void actionPerformed(ActionEvent e)

**3.6 *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.
  + java.awt.BorderLayout: A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.
  + java.awt.Color: The Color class is used encapsulate colors in the default RGB color space or colors in arbitrary color spaces identified by a [ColorSpace](http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/color/ColorSpace.html).
  + java.awt.Dimension: This encapsulates the width and height of a component (in integer precision) in a single object.
  + java.awt.GridLayout: The GridLayout class is a layout manager that lays out a container's components in a rectangular grid for a better view of all the buttons and textfields which are added to the main panel.
* import java.awt.event
  + java.awt.event.ActionEvent;
  + java.awt.event.ActionListener;
* 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.
  + javax.swing.JButton;
  + javax.swing.JFrame; javax.swing.JLabel; javax.swing.JPanel;

***3.7 Runnable Interface***

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behaviour of the class.

In this program for each queues from the array of queues declared in the Shop class, we declare a thread. This means that we will have a run method in the class Queue which will determin whether or not to execute an action depending on the time. In this case the queue will remain unchanged while a client is being served and only after this time has passed the client is removed from the queue. This is done with the following instructions:

try {

Thread.sleep(time);

}

catch (InterruptedException e) {}

clienti.remove();

***3.8 User Interface***

When running the application, the window will open and it will provide to the user the possibility of giving inputs and choosing what operation he likes to be executed. This window is constructed in the Gui class using some predefined classes and instructions.

The user interface is based on the properties of the above mentioned packages. All the objects we need are declared as attributes of the Gui class and they are initialized in the constructor of this class. For executing the operation commanded by the user we use the predefined functions from the ”ActionListener” interface.

The ActionListener functions

The listener interface is for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked. In this case the only events that occur are when the user clicks on one of the operation buttons from the graphical interface.

We take as an example the instructions that need to be executed when clicking on the “P(val)” button, which leads to the determination and displaying the value of the polynomial written in ”First Polynomial” field in point val. For this we need another class which implements the ActionListener:

private class StartListener implements ActionListener

This class will contain the method which executes all the instruction needed in order to fulfill the selected operation.

public void actionPerformed (ActionEvent e)

***4. Using and testing the application***

The application was tested with the JUnit classes:

import org.junit.Test;

import static org.junit.Assert.assertEquals;

In order to use the application open Homework4.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 buttons.

***5.Conclusions***

Achieving such a program may be hard both in terms of algorithms, graphical structure.

For a better performance there should be implemented all cases where exceptions can occur and the application stops working due to an error made ​​by the user. Also, the division method should be done so it can calculate the reminder as well. Another thing that could be improved is the display so that it would be more elegant.

***6.References***

[*http://users.utcluj.ro/~jim/OOPE/*](http://users.utcluj.ro/~jim/OOPE/)

[*http://docs.oracle.com/javase/7/docs/api/overview-summary.html*](http://docs.oracle.com/javase/7/docs/api/overview-summary.html)

[*http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html*](http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/javadoc.html)

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