Programowanie w jezyku JAVA
Lab 1: Podstawowe elementy jezyka w przykładzie

Piotr Szuster

8 marca 2022

1 Definicja modelowanego problemu

Rozważmy równanie nastepujacej postaci:

$$ax^2 + bx + c = 0 \tag{1}$$

W zależności od wartości przyjetych współczynników a, b, c równanie to może mieć różna liczbe rozwiazań. Dla $a \neq 0$ liczba rozwiazań określana jest przez wyróżnik trójmianu kwadratowego:

$$\Delta = b^2 - 4ac \tag{2}$$

Równanie (1) posiada dwa rozwiazania $x_0, x_1 \in R$ gdy $\Delta > 0$:

$$x_0 = \frac{-b - \sqrt{\Delta}}{2a} \quad x_1 = \frac{-b + \sqrt{\Delta}}{2a} \tag{3}$$

Równanie (1) posiada jedno rozwiazanie $x_0 \in R$ gdy $\Delta = 0$:

$$x_0 = \frac{-b}{2a} \tag{4}$$

Równanie (1) posiada dwa rozwiazania $x_0, x_1 \in C$ gdy $\Delta < 0$:

$$x_0 = \frac{-b - i\sqrt{-\Delta}}{2a} \quad x_1 = \frac{-b + i\sqrt{-\Delta}}{2a} \tag{5}$$

gdzie i to jednostka urojona. Dla $a = 0, b \neq 0$ równanie przyjmuje postać

$$bx + c = 0 \tag{6}$$

posiadajac jedno rozwiazanie x_0:

$$x_0 = \frac{-c}{b} \tag{7}$$

Dla $a = 0, b = 0, c \neq 0$ może być sprzeczne, badź posiadać nieskończenie wiele rozwiazań.

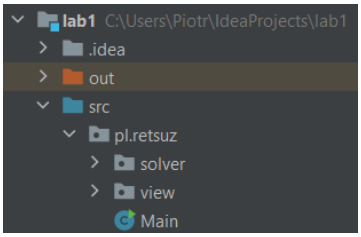
2 Zadanie

Zaimplementować program w jezyku JAVA, który po wprowadzeniu przez użytkownika parametrów a,b,c wyznaczy rozwiazania równania (1), obejmujac wszystkie przypadki wymienione w sekcji 1. Program ma wyświetlić wyniki na ekranie. Wykorzystać interfejs wejścia/wyjścia konsoli.

3 Rozwiazanie zadania

3.1 Struktura projektu

Aby właściwie rozwiazać problem, pozostajac w zgodzie z dobrymi praktykami programowania obiektowego, postawiony w ramach laboratorium należy podzielić rozwiazanie na paczki, które beda zawierać klasy powiazane ze soba. Zgodnie z treścia zadania można wyszczególnić dwa obszary, które obejma rozwiazanie problemu: obliczenie rozwiazań równania oraz komunikacje miedzy użytkownikiem i aplikacja. W zwiazku z powyższym wyszczególniono paczki: solver oraz view. solver jest paczka, która zawierać bedzie klasy oraz interfejsy zwiazane z obliczeniami, a view odpowiedzialna za komunikacje człowieka z komputerem. Struktura całego projektu przedstawiona jest na rysunku 1.



Rysunek 1: Struktura projektu

4 Interfejsy

Klasyczna definicja interfejsu jest określona jako grupa powiazanych z soba metod. Interfejs zawiera jedynie deklaracje metod. Implementacja interfejsu przez klase wymusza implementacje metod deklarowanych w interfejsie. Celem wykorzystania interfejsów jest ukrycie szczegółowej implementacji przez ekspozycje tylko tych metod, które sa niezbedne do komunikacji z pozostałymi klasami. Pierwszy zdefiniowany interfejs to QFormulaSolver w paczce solver 2.

```
package pl.retsuz.solver;

public interface QFormulaSolver {
    public double[] solve();
    public void setInitialParameters(double a, double b, double c);
    public boolean IsComplex();

}
```

Rysunek 2: Interfejs QFormulaSolver

metoda solve() bedzie odpowiedzialna za rozwiazanie równania. metoda setInitialParameters() za ustawienie współczynników równania, a metoda IsComplex() do zwrócenia informacji o tym czy obliczone wyniki sa sprecyzowane dla liczb zespolonych.

Drugi interfejs odpowiada zawiera deklaracje metod, które odpowiadać beda za komunikacje człowiek komputer 3.

```java
package pl.retsuz.view;

import pl.retsuz.solver.QFormulaSolver;

public interface QuadraticEquationSolverView {
    public void View();
    public void Init(QFormulaSolver solver);
}
```

Rysunek 3: QuadraticEquationSolverView

View() odpowiadać bedzie za interakcje interfejsem konsolowym, a Init() za przekazanie informacji o referencji do obiektu klasy implementujacej interfejs QFormulaSolver. Należy zwrócić uwage, że do wykorzystania interfejsu QFormulaSolver konieczne było wykonanie odpowiedniego importu.

5 Implementacja obliczeń

W paczce solver należy utworzyć klase odpowiedzialna za implementacje metod interfejsu QFormulaSolver. Klasa jest pojeciem podstawowym paradygmatu obiektowego programowania i jest rozumiana jako definicja dopuszczalnej tożsamości, stanu oraz zachowania obiektów. Klasa ta ma posiadać nazwe QuadraticFormulaSolver 4.

```java
package pl.retsuz.solver;

public class QuadraticFormulaSolver implements QFormulaSolver {

    private double a;
    private double b;
    private double c;
    private boolean isComplex;

    public QuadraticFormulaSolver(){
        a=0;
        b=0;
        c=0;
    }

    public void setInitialParameters(double a, double b, double c){
        this.a=a;
        this.b=b;
        this.c=c;
    }

    public boolean IsComplex() { return isComplex; }
```

Rysunek 4: QuadraticFormulaSolver cz. 1

Warto zuważyć, że wymuszenie interfejsu odbyło sie za pomoca użycia implements. Wymusiło to dalsza implementacje metod. W jezyku Java nie ma wielodziedziczenia, a klasy moga implementować wiele interfejsów. Tożsamość obiektów klasy QuadraticFormulaSolver obejmuje ich przynależność do klasy QuadraticFormulaSolver oraz implementacje interfejsu QFormulaSolver. Dopuszczalny stan obiektów tej klasy definiuja cztery zmienne z prywatnym modyfikatorem dostepności, a zachowanie zestaw metod. Konstruktor bezargumentowy wykorzystano do wstepnego ustalenia wartości współczynników. Metoda setInitialParameters musiała zostać wykorzystania do właściwej inicjalizacji ze wzgledu na wykorzystanie interfejsu (interfejsy nie moga deklarować konstruktorów).

```
31      protected double delta() { return b*b-4*a*c; }
34
35      protected boolean isQuadratic(){
36          return a != 0;
37      }
38      protected double[] twoRoots(double d){
39          double sd = Math.sqrt(d);
40          double[] result=new double[2];
41          result[0] = (-b-sd)/2*a;
42          result[1]= (-b+sd)/2*a;
43          return result;
44      }
45
46      protected double[] singleRoot(){
47          double [] result = new double[1];
48          result[0]=-b/2*a;
49          return result;
50      }
51
52      protected double[] solveQuadratic(){
53          isComplex=false;
54          double d = this.delta();
55          if(d>0) return twoRoots(d);
56          else if (d==0) return singleRoot();
57          else {
58              isComplex=true;
59              return twoRoots(-d);
60          }
61      }
```

Rysunek 5: QuadraticFormulaSolver cz. 2

Rysunek 5 ukazuje elementarne funkcje, które pozwalaja właściwie zdekomponować problem do postaci prostych obliczeń matematycznych. Aby zachować ogólność formy przekazu danych wykorzystano tablice dwuelementowe.

```
63      protected double solveLinear() throws SolverException {
64          if(b!=0) return -c/b;
65          else{
66              if(c==0) throw new InfiniteRootsException();
67              else throw new ContradictoryEquationException();
68          }
69      }
70      public double[] solve() throws SolverException {
71          if(isQuadratic()) return solveQuadratic();
72          else{
73              double[] result =new double[1];
74              result[0]=solveLinear();
75              return result;
76          }
77      }
78  }
```

Rysunek 6: QuadraticFormulaSolver cz. 3

Rysunek 6 ukazuje dwie funkcje, które wykorzystuja wyjatki do sygnalizacji sytuacji szczególnych. Wyjatki stanowia mechanizm sterowania przepływem - kontroli działania programu i zwykle sa używane

do sygnalizowania błedów. W omawianym przypadku metoda solveLinear() rzuca (throws) dedykowany wyjatek SolverException, który został zdefiniowany w osobnej klasie. Posiada on podklasy InfiniteRootsException() oraz ContradictoryEquationException(). Metoda solve wykorzystujac poprzednio zdefiniowane funkcje, scala wyniki obliczeń obejmujac wszystkie przypadki.

5.1 Dedykowane wyjatki

Aby wykorzystać wyjatki w pierwszej kolejności nalezy utworzyć paczke exceptions w obrebie wewnatrz paczki solver. W tej paczce nalezy utworzyć trzy klasy.

```java
package pl.retsuz.solver.exceptions;

public class SolverException extends Exception{
    public SolverException(String message){
        super(message);
    }
}
```

```java
package pl.retsuz.solver.exceptions;

public class InfiniteRootsException extends SolverException{
    public InfiniteRootsException(){
        super("Równanie ma nieskończenie wiele rozwiązań!");
    }
}
```

```java
package pl.retsuz.solver.exceptions;

public class ContradictoryEquationException extends SolverException{
    public ContradictoryEquationException(){
        super("Równanie sprzeczne!");
    }
}
```

Rysunek 7: Dedykowane wyjatki.

SolverException rozszerza ogólna klase wyjatków jezyka Java: Exception. Owo rozszerzenie odbywa sie przez wykorzystanie extends i w swej istocie realizuje mechanizm dziedziczenia - relacji przekazywania cech miedzy klasami. Dziedziczeniu podlegaja wszystkie cechy: tożsamość, stan i zachowanie. Odwołanie do konstruktora klasy bazowej realizowane jest przez użycie metody super.

6 Komunikacja człowiek-komputer

Komunkacja człowieka z komputerem odbywa sie za pomoca klasy QuadraticEquationSolverConsoleView. Klasa ta implementuje interfejs QuadraticEquationSolverView. Zawiera ona dwa prywatne pola, przechowujace referencje (adresy pamieci) kolejno do obiektów odpowiedzialnych za rozwiazywanie równania kwadratowego oraz za parsowanie typów prostych oraz ciagów znakowych z wykorzystaniem wyrażeń regularnych (java.util.Scanner).

```java
package pl.retsuz.view;

import pl.retsuz.solver.QFormulaSolver;
import pl.retsuz.solver.exceptions.SolverException;
import java.util.Scanner;

public class QuadraticEquationSolverConsoleView implements QuadraticEquationSolverView {
    private QFormulaSolver solver;
    private Scanner sc;

    protected double parseWithMessage(String message){
        System.out.println(message);
        String line;
        double res;
        try {
            line = sc.nextLine();
            res=Double.parseDouble(line);

        }catch (Exception ex){
            System.err.println("Wprowadzono niepoprawne dane!");
            res= parseWithMessage(message);
        }
        return res;
    }
```

Rysunek 8: Wyświetlanie cz. 1

Metoda parseWithMessage() odpowiada za wyświetlenie komunikatu informujacego użytkownika o tym jaki parametr ma wprowadzić (System.out.println()), pobranie danych z klawiatury (nextLine()), parsowanie (Double.parseDouble()) oraz kontrole poprawności wprowadzanych danych przy pomocy wyjatków. Wyjatki rzucane przez metode parseDouble() sa łapane przez catch w bloku try by wyświetlić stosowny komunikat w strumieniu err. Metoda parseWithMessage() została zaimplementowana w taki sposób aby można ja było ponownie wykorzystywać.

Metoda parseFactors() za pomoca wywołań parseWithMessage() wymusza wprowadzenie danych o odpowiednich współczynnikach oraz inicjalizuje za pomoca metody setInitialParameters() obiekt solvera. Metoda displaySolutions() po przyjeciu tablicy wyników formatuje stosowny ciag znaków tak aby podać dziedzine oraz liczbe rozwiazań.

```java
    protected void parseFactors(){
        double a,b,c;
        a = parseWithMessage("Wprowadź współczynnik a:" );
        b = parseWithMessage("Wprowadź współczynnik b: ");
        c = parseWithMessage("Wprowadź współczynnik c: ");
        this.solver.setInitialParameters(a,b,c);
    }

    protected void displaySolutions(double [] res){
        String label = "Podane równanie posiada następujące rozwiązania w dziedzinie liczb ";
        if(solver.IsComplex())label += "zespolonych:\n";
        else label+= "rzeczywistych:\n";
        for(int i=0;i<res.length;i++) label+="x"+i+"="+res[i]+";\t";
        System.out.println(label);
    }

    protected void getSolution(){
        double [] res;
        try{
            res=solver.solve();
            displaySolutions(res);
        }
        catch(SolverException ex){
            System.err.println(ex.getMessage());
        }
    }
```

Rysunek 9: Wyświetlanie cz. 2

getSolution() definiuje ciag obliczeń i wyświetlenia wyników oraz obsługe sytuacji szczególnych (nieskonczenie wiele rozwiazań, sprzeczność). View() w petli nieskończonej pyta użytkownika o parametry kolejnych równań oraz podaje wyniki za pomoca funkcji składowej, poprzednio omówionej.

Init() inicjuje klase. Adnotacja @Override uwidacznia mechanizm przeciażenia metod wirtualnych z interfejsu (tych które sa przeznaczone do przeciażenia). Przeciażenie polega na ponownej definicji ciała

metody, a w przypadku metody abstrakcyjnej definicja ciała jest wymuszona (metoda abstrakcyjna ciała nie posiada).

```
52      @Override
53      public void View() {
54          while(true) {
55              parseFactors();
56              getSolution();
57          }
58      }
59
60      @Override
61      public void Init(QFormulaSolver solver) {
62          this.solver=solver;
63          this.sc = new Scanner(System.in);
64      }
65  }
```

Rysunek 10: Wyświetlanie cz. 3

7 Program w całości

```
1   package pl.retsuz;
2
3   import pl.retsuz.solver.QFormulaSolver;
4   import pl.retsuz.solver.QuadraticFormulaSolver;
5   import pl.retsuz.view.QuadraticEquationSolverConsoleView;
6   import pl.retsuz.view.QuadraticEquationSolverView;
7
8   public class Main {
9       static QFormulaSolver solver;
10      static QuadraticEquationSolverView view;
11      public static void main(String[] args) {
12          solver= new QuadraticFormulaSolver();
13          view=new QuadraticEquationSolverConsoleView();
14          view.Init(solver);
15          view.View();
16      }
17  }
```

Rysunek 11: Klasa main.

Klasa main zawiera dwa statyczne pola, które przechowuja referencje do obiektów klas implementujacych interfejsy QFormulaSolver oraz QuadraticEquationSolverView. W statycznej metodzie main pola te sa inicjalizowane za pomoca konstruktorów. W dalszej cześci obiekt klasy odpowiedzialny za komunikacje ma przypisana referencje do obiektu klasy odpowiedzialnej za obliczenia.

7.1 Podsumowanie

Na co pozwala wykorzystanie interfejsów? W jaki sposób teoretycznie można rozszerzyć program o obsługe interfesju graficznego? W jaki sposób można byłoby zmodyfikowac interfejsy aby umożliwić obsługe dowolnego wielomianu? Na co pozwala wykorzystanie wielu funkcji o niewielkich ciałach?