**4.1. Polimorfizmul**

**4.1.1. Metodele statice şi polimorfismul**

Pentru metodele statice nu se aplică principiile polimorfismului. La utilizarea referinţei pentru accesul la membrul static, compilatorul, la alegerea metodei sau obiectului, ţine cont de tipul referinţei, dar nu tipul obiectului, la care se referă.

/\* exemplu # 1: comportarea metodelor statice: \*/

**class** StaticA {

**public** **static** **void** show(){

System.out.println(

"metoda show() din StaticA");

}}

**class** StaticB **extends** StaticA {}

**class** StaticC **extends** StaticB {

**public** **static** **void** show(){

System.out.println("

metoda show() din StaticC");

}}

**public** **class** StaticDemo {

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

StaticA s1 = **new** StaticC();

StaticB s2 = **new** StaticC();

StaticC s3 = **new** StaticC();

s1.show();

s2.show();

s3.show();

}}

Rezultatul realizării programului:

**metoda show() din StaticA**

**metoda show() din StaticA**

**metoda show() din StaticC**

La iniţializarea obiectelor s1 şi s2, metoda **show()**, este apelata de clasele de bază **StaticA** şi **StaticB**. Polimorfismul se observă numai la moştenirea metodei. Pentru obiectul s3 va fi apelata metoda proprie **show(),** care rezultă din modul de declarare. Dacă specificatorul **static** de extras din declaraţia metodei, atunci apelarea metodelor vor fi efectuate în conformitate cu principiile polimorfismului.

**4.2. Clasa Object**

La baza ierarhiei claselor stă clasa **Object**. Obiectul de tip **object** se poate referi la orice obiect din altă clasă, în plus, o variabilă de tip **object** se poate referi la orice tablou, deoarece tablourile sunt realizate ca clase. În clasa **Object se** defineşte un set de metode, care sunt moştenite de toate clasele. Metoda **equals()**, la compararea a două obiecte întoarce **true,** daca obiectele sunt echivalente, şi **false** - in caz contrar. Dacă este nevoie de comparat obiectele clasei create, această metodă necesită să fie suprascrisă in clasa dată. Metoda **toString()** returnează un şir cu descrierea obiectului in forma:

**getClass().getName()+'@' +Integer.toHexString(hashCode())**

Metoda se apelează în mod automat, atunci când un obiect este afişat prin metodele **println(), print()**, şi altele. Atunci când se creează o clasă este recomandat să suprascrie metoda **toString()**, pentru a aduce la tipul de obiect nou creat.

/\* Exemplul # 2: suprascrierea metodelor equals() şi toString() : \*/

**class** Point {

**protected** **byte** b;

**protected** String str;

**public** Point(**byte** n, String s) {

b = n;

str = s;}

**public** Point() {

**this**((**byte**)0, "NoName");}

**public** **boolean** equals(Object obj) {

**if** (obj **instanceof** Point)

**return** (**this**.b == ((Point) obj).b) && (str.equals(((Point) obj).str));

**return** **false**;}

**public** String toString() {

**return** getClass().getName() + "@"+ " name=" + str + " b=" + b;

}}

**class** PointZ **extends** Point{

**short** s = 100;

}

Metoda **equals ()** se suprascrie pentru clasa **Point**, în aşa fel încât să se asigure că obiectul primit este un obiect de tip **Point** sau unul din moştenitorii săi, precum şi pentru a compara conţinutul obiectelor **b** şi **str**, respectiv, obiectului apelat şi obiectului transmis. Metoda **toString()** se suprascrie în aşa fel ca, pe lângă informaţia despre pachetul standard in care se conţine clasa **Point** şi implicit numele clasei, să afişeze valorile câmpului obiectului care apelează această metodă, în loc de codul de distribuire, cum se face în clasa **Object.**

Are loc apelul unui constructor dintr-un alt constructor, cu transmiterea parametrilor, precum şi conversia de tip **int** la tipul **byte**, aceasta conversie nu este efectuată implicit, deoarece este posibila pierderea informaţiei.

/\* exemplul # 3 : utilizarea metodelor equals() şi toString(: \*/

**package** com.mypack;

**public** **class** PointDemo {

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

Point p1 = **new** Point((**byte**) 1, "Salut");

Point p2 = **new** Point((**byte**) 1, "Salut");

PointZ p3 = **new** PointZ();

Point p4 = **new** Point();

System.out.println(p1.equals(p2));

System.out.println(p1.equals(p3));

System.out.println(p4.equals(p3));

System.out.println(p3.equals(p4));

System.out.println(p1.toString());

}}

Rezultatul realizării programului:

**true**

**false**

**true**

**true**

**com.mypack.Point@ name=Salut b=1**

Suprascrierea în acest mod a metodei **equals()** permite să compare obiectele clasei de bază cu obiectele claselor derivate, dar numai pentru acele cîmpuri care sunt comune.

**4.3. Distrugerea obiectelor**

Deoarece obiectele sunt create dinamic cu ajutorul operatorului **new**, este de dorit de a cunoaşte mecanismul de distrugere a obiectelor şi o modalitate de eliberare a memoriei pentru redistribuirea de mai târziu. Java automat face eliberarea memoriei, ocupată de obiect, prin intermediul mecanismului special. Atunci când nici un link la obiect nu există, adică link-ul la el a ieşit din domeniul de aplicare a programului, se presupune că obiectul nu mai este necesar, şi memoria ocupată de obiect poate fi eliberată. Distrugerea obiectelor are loc în mod neregulat în timpul execuţiei programului.

Uneori, un obiect are nevoie pentru a efectua o acţiune înainte de a elibera memorie. De exemplu, pentru a elibera resurse externe. Pentru a gestiona astfel de situaţii, se foloseşte mecanismul **finalization**. Pentru a utiliza **finalization**, trebuie de definit metoda **finalize().** Maşină virtuală activează această metodă atunci cînd necesită distrugerea obiectului clasei. În interiorul metodei **finalize()** trebuie de determinat acţiunile, care trebuie îndeplinite pînă la distrugerea obiectului. Înainte de eliberarea memoriei pentru obiectul dat se apelează metoda **finalize().**

Metoda **finalize()** are forma:

**protected void finalize(){**

**//** codul de finalizare**}**

Nivelul de acces **protected** interzice accesul la metoda **finalize()** codurilor, definite în afara clasei date. Metoda **finalize()** este apelata chiar înainte de eliberarea memoriei, şi nu atunci când obiectul dispare din cîmpurile de identificare deoarece este imposibil să se determine, cînd **finalize()** este executat. Această metodă poate să nu fie, în general, executata.

/\*exemplul # 4: distrugerea obiectului\*/

**class** Demo {

**private** **int** a;

**public** Demo(**int** a) {

**this**.a = a;}

**protected** **void** finalize() {

System.out.println("obiectul este distrus, a=" + a);}}

**public** **class** FinalizeDemo {

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

Demo d1 = **new** Demo(1);

d1 = **null**;

Demo d2 = **new** Demo(2);

Object d3 = d2; //1

//Object d3 = new Demo(3); //2

d2 = d1;

System.gc();

// metodă pentru distrugerea obiectului

}}

Ca urmare a executării acestui cod înainte de apelarea metodei **gc()**, fără referinţă rămâne doar un obiect.

**obiectul este distrus, a=1**

Dacă de comentat linia 1 şi de decomentat linia 2, înainte de executarea metodei **gc()** referinţele au pierdut două obiecte.

**obiectul este distrus, a=1**

**obiectul este distrus, a=2**

**Lucrare de laborator nr. 4**

**1. Tema lucrării:**

Polimorfizmul.

**2.**  **Scopul lucrării:**

* Însuşirea modalităţilor de realizare a polimorfizmului în Java;

**3. Etapele de realizare:**

1. Crearea clase noi;
2. Crearea metodelor necesare;
3. Realizarea formelor de polimorfizm;
4. Crearea interfeţii programului;
5. Prezentarea lucrării.

**4. Exemplu de realizare:**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

public class Prob4 {

public static void main(String[] args) {

Complex tt = new Complex(1,6); Complex tttt= (Complex) tt.def

(new Complex(2,-1));

System.out.println(tttt);

ComplexPolynom pol = new ComplexPolynom(5);

for (int i= 0;i<pol.length;i++){ pol.setComplexPolynom(i,

(int)(Math.random()\*10), (int)(Math.random()\*10), (int)(Math.random()\*10));

}

ComplexPolynom pol2 = new ComplexPolynom(3);

for (int i= 0;i<pol2.length;i++) { pol2.setComplexPolynom(i, (int)(Math.random()\*10), (int)(Math.random()\*10), (int)(Math.random()\*10));

}

ComplexPolynom pol1 = (ComplexPolynom)pol.def(pol2); System.out.println(pol.toStringAll()); System.out.println(pol2.toStringAll()); System.out.println(pol1.toStringAll());

ComPolFram fram = new ComPolFram();

fram.setVisible(true);

}}

class ComPolFram extends JFrame{

public ComPolFram(){

this.setTitle("Complex Poligon"); this.setDefaultCloseOperation(

JFrame.EXIT\_ON\_CLOSE);

this.setSize(600, 500);

this.setLocation(300, 200);

Box Hbox1 = Box.createHorizontalBox();

Hbox1.add(new JLabel("n="));

nn = new JTextField(10); nn.setMaximumSize(nn.getPreferredSize());

Hbox1.add(nn);

butmac("add",Hbox1);

butmac("+",Hbox1);

butmac("-",Hbox1);

Hbox2 = Box.createHorizontalBox();

Hbox3 = Box.createHorizontalBox();

Hbox4 = Box.createHorizontalBox();

Hbox5 = Box.createHorizontalBox();

Hbox6 = Box.createHorizontalBox();

Box Vbox = Box.createVerticalBox();

Vbox.add(Hbox1);

Vbox.add(Hbox2);

Vbox.add(Hbox3);

Vbox.add(Hbox4);

Vbox.add(Hbox5);

Vbox.add(Hbox6);

add(Vbox,BorderLayout.CENTER);

//this.pack();

}

private JTextField nn;

private JTextField[] n1,n2,n3;

private JTextField[] m1,m2,m3;

private Box Hbox2,Hbox3,Hbox4,Hbox5,Hbox6;

private void butmac(String s,Box b){

JButton but = new JButton(s);

but.addActionListener(

new ActionListener(){

public void actionPerformed(ActionEvent e){ System.out.println(e.getActionCommand());

//Hbox2 = Box.createHorizontalBox();

int n = Integer.parseInt(nn.getText());

if (e.getActionCommand()=="add"){

Hbox2.removeAll();

Hbox3.removeAll();

n1 = new JTextField[n];

n2 = new JTextField[n];

n3 = new JTextField[n];

m1 = new JTextField[n];

m2 = new JTextField[n];

m3 = new JTextField[n];

for(int i=0;i<n;i++){

n1[i] = new JTextField(3); n1[i].setMaximumSize(nn.getPreferredSize());

n2[i] = new JTextField(3); n2[i].setMaximumSize(nn.getPreferredSize());

n3[i] = new JTextField(3); n3[i].setMaximumSize(nn.getPreferredSize());

m1[i] = new JTextField(3); m1[i].setMaximumSize(nn.getPreferredSize());

m2[i] = new JTextField(3); m2[i].setMaximumSize(nn.getPreferredSize());

m3[i] = new JTextField(3); m3[i].setMaximumSize(nn.getPreferredSize());

Hbox2.add(new JLabel("(r="));

Hbox2.add(n1[i]);

Hbox2.add(new JLabel("i="));

Hbox2.add(n2[i]);

Hbox2.add(new JLabel("b="));

Hbox2.add(n3[i]);

Hbox2.add(new JLabel(") "));

Hbox3.add(new JLabel("(r="));

Hbox3.add(m1[i]);

Hbox3.add(new JLabel("i="));

Hbox3.add(m2[i]);

Hbox3.add(new JLabel("b="));

Hbox3.add(m3[i]);

Hbox3.add(new JLabel(") "));

}

pack(); }

else

if(e.getActionCommand()=="+" || e.getActionCommand()=="-")

{ Hbox4.removeAll();

Hbox5.removeAll();

Hbox6.removeAll(); ComplexPolynom pol1 = new ComplexPolynom(n);

for (int i=0;i<n;i++){

pol1.setComplexPolynom(i, Double.parseDouble(n1[i].getText()), Double.parseDouble(n2[i].getText()), Double.parseDouble(n3[i].getText()));

}

ComplexPolynom pol2 = new ComplexPolynom(n);

for (int i=0;i<n;i++){

pol2.setComplexPolynom(i, Double.parseDouble(m1[i].getText()), Double.parseDouble(m2[i].getText()), Double.parseDouble(m3[i].getText()));

}

ComplexPolynom pol3;

if(e.getActionCommand()=="+"){

pol3 = (ComplexPolynom)pol1.adun(pol2);

}

else{

pol3 = (ComplexPolynom)pol1.def(pol2);

}

Hbox4.add(

new JLabel("("+pol1.toStringAll()+")"));

Hbox5.add(

new JLabel("+("+pol2.toStringAll()+")"));

Hbox6.add(

new JLabel("=("+pol3.toStringAll()+")"));

pack();

}}});

b.add(but);

}}

abstract class Scadere {

abstract public Object adun(Object t);

abstract public Object invers();

public Object def(Object t) {

Scadere tt = (Scadere)t;

return adun(tt.invers());

}}

class Complex extends Scadere{

public Complex() {

real = 0;

img = 0;

}

public Complex(double Real,double Img) {

real = Real;

img = Img;

}

public void setComplex(double Real,

double Img) {

real = Real;

img = Img;

}

public String toString(){

return real+(img>0 ? "+" : "")+(img!=0 ? img+ "i" : "");

}

public Complex adun(Object t) {

Complex tt = (Complex)t;

Complex ret = new Complex();

ret.img = img + tt.img;

ret.real = real + tt.real;

return ret;

}

public Complex clone() {

Complex temp = new Complex();

temp.img = img;

temp.real = real;

return temp;

}

public Complex invers() {

Complex temp = this.clone();

temp.img = temp.img\*(-1);

temp.real = temp.real\*(-1);

return temp;

}

protected double real,img;

}

class ComplexPolynom extends Complex {

public ComplexPolynom() {

super();

pol = new ComplexPolynom[1];

pol[0] = this;

baz = 0;put = 0;

length = 1;

}

public ComplexPolynom(int n) {

super();

pol = new ComplexPolynom[n];

pol[0] = this;

for(int i=1;i<pol.length;i++){

pol[i] = new ComplexPolynom();

}

length = n;

}

public void setComplexPolynom(

int n,double Real,double Img,double Baz ) {

pol[n].baz = Baz;

pol[n].put = n;

pol[n].setComplex(Real, Img);

}

public ComplexPolynom getComplexPolynom(int n) {

return pol[n];

}

public String toString(){

return (baz>=0 ? "+":"")+baz+"("+super.toString()+")^"+put;

}

public String toStringAll() {

String output = "";

for(ComplexPolynom temp:pol) {

output+=temp;

}

return output;

}

public ComplexPolynom adun(Object t) {

ComplexPolynom tt = (ComplexPolynom)t;

int maxl = tt.length>length?tt.length:length;

ComplexPolynom ret =

new ComplexPolynom(maxl);

for(int i=0;i<maxl;i++) {

if (tt.length>i){

ret.pol[i].baz = tt.pol[i].baz;

ret.pol[i].img = tt.pol[i].img;

ret.pol[i].real = tt.pol[i].real;

}

if (length>i){

ret.pol[i].baz += pol[i].baz;

ret.pol[i].img += pol[i].img;

ret.pol[i].real += pol[i].real;

}

ret.pol[i].put = i;

}

return ret;

}

public ComplexPolynom invers() {

ComplexPolynom temp =

new ComplexPolynom(length);

for(int i=0;i<length;i++) {

temp.pol[i].baz = pol[i].baz\*(-1);

temp.pol[i].put = pol[i].put;

temp.pol[i].img = pol[i].img\*(-1);

temp.pol[i].real = pol[i].real\*(-1);

}

return temp;

}

public ComplexPolynom clone() {

ComplexPolynom temp =

new ComplexPolynom(length);

for(int i=0;i<length;i++) {

temp.pol[i].baz = pol[i].baz;

temp.pol[i].put = pol[i].put;

temp.pol[i].img = pol[i].img;

temp.pol[i].real = pol[i].real;

}

return temp;

}

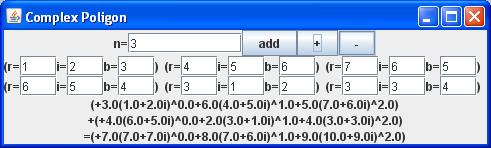
public int length;

private double baz,put;

private ComplexPolynom[] pol;

}

Rezultatul realizării programului:



**5. Probleme propuse:**

Realizarea polimorfismului pentru clasele date cu crearea metodelor conform specificului claselor.

1.      Fraction 🡨 ComplexFracion.

2.      Complex 🡨 ComplexPolynom.

1. Fraction 🡨 FracionArray.

4.      Fraction 🡨 FracionPolynom.

5.      Complex 🡨 ComplexArray.

6.      Complex 🡨 ComplexFraction.

7.      Polynom 🡨 ComplexPolynom.

1. Polynom 🡨 FractionPolynom.
2. Polinom <- PoliniomArray
3. Array <-ComplexArray
4. Array<-FractionArray
5. Array <-ComplexFraction