Nicoleta Radu, Tema c++

TEMA 9 – POLIMORFISM

# 1

#pragma once

#include <iostream>

class Pereche

{

protected:

// Protected Variables

double firstOperand{}, secondOperand{};

// Protected Functions

void getOperands(double&, double&);

void setOperands(double, double);

private:

// Private functions

void initVariables();

public:

// Constructors & Destructor

Pereche();

Pereche(double,double);

Pereche(Pereche&);

~Pereche();

// Functions

void display();

};

#include "Pereche.h"

void Pereche::getOperands(double& \_firstOp, double& \_secondOp)

{

\_firstOp = firstOperand;

\_secondOp = secondOperand;

}

void Pereche::setOperands(double \_firstOp, double \_secondOp)

{

firstOperand = \_firstOp;

secondOperand = \_secondOp;

}

void Pereche::initVariables()

{

firstOperand = 0.0;

secondOperand = 0.0;

}

Pereche::Pereche()

{

initVariables();

}

Pereche::Pereche(double \_firstOperand, double \_secondOperand): firstOperand(\_firstOperand), secondOperand(\_secondOperand)

{

}

Pereche::Pereche(Pereche& aux)

{

aux.firstOperand = firstOperand;

aux.secondOperand = secondOperand;

}

Pereche::~Pereche()

{

}

void Pereche::display()

{

/\*

Display Complex numbers

Prints firstOperand and secondOperand

\*/

std::cout << firstOperand << "," << secondOperand << std::endl;

}

INHERITED

#pragma once

#include "Pereche.h"

class Complex: public Pereche

{

private:

// Private functions

Complex negativ();

Complex conjugat();

double absolut();

public:

// Constructors and Destructor

Complex();

Complex(double,double);

~Complex();

// Functions

friend Complex operator+(const Complex, const Complex);

friend Complex operator-(const Complex, const Complex);

friend Complex operator\*(const Complex,const Complex);

friend Complex operator/(Complex,Complex);

};

#include "Complex.h"

#include <cmath>

Complex::Complex() : Pereche()

{

// No args constructor

}

Complex::Complex(double \_firstOp, double \_secondOp) : Pereche(\_firstOp, \_secondOp)

{

// Overloaded constructor

}

Complex::~Complex()

{

// Destructor

}

Complex Complex::negativ()

{

/\*

Makes operands negative numbers

Returns@ Complex number

\*/

Complex rezultat;

rezultat.firstOperand = -firstOperand;

rezultat.secondOperand = -secondOperand;

return rezultat;

}

Complex Complex::conjugat()

{

/\*

Makes conjugate for both operands

Returns@ Complex number

\*/

Complex rezultat;

rezultat.firstOperand = firstOperand;

rezultat.secondOperand = -secondOperand;

return rezultat;

}

double Complex::absolut()

{

/\*

Makes the absolute values of the two operands

Returns@ Complex number

\*/

return sqrt((firstOperand \* firstOperand) + (secondOperand \* secondOperand));

}

Complex operator+(const Complex a, const Complex b)

{

/\*

Overloaded '+' operator calculates sum of two

Complex numbers

Returns@ Complex number

\*/

Complex rezultat;

rezultat.firstOperand = a.firstOperand + b.firstOperand;

rezultat.secondOperand = a.secondOperand + b.secondOperand;

return rezultat;

}

Complex operator-(const Complex a, const Complex b)

{

/\*

Overloaded '-' operator calculates difference of two

Complex numbers

Returns@ Complex number

\*/

Complex rezultat;

rezultat.firstOperand = a.firstOperand + (-b.firstOperand);

rezultat.secondOperand = a.secondOperand + (-b.secondOperand);

return rezultat;

}

Complex operator\*(const Complex a , const Complex b)

{

/\*

Overloaded '\*' operator calculates the multiplications of two

Complex numbers

Returns@ Complex number

\*/

Complex rezultat;

rezultat.firstOperand = a.firstOperand \* b.firstOperand - a.secondOperand \* b.secondOperand;

rezultat.secondOperand = a.firstOperand \* b.secondOperand - a.secondOperand \* b.firstOperand;

return rezultat;

}

Complex operator/( Complex a, Complex b)

{

/\*

Overloaded '/' operator calculates the division of

two Complex numbers

Returns@ Complex number

\*/

Complex rezultat;

Complex n;

rezultat.firstOperand = n.firstOperand / pow(a.absolut(), 2);

rezultat.firstOperand = n.secondOperand / pow(a.absolut(), 2);

return rezultat;

}

INHERITED

#pragma once

#include "Pereche.h"

#include <iostream>

class Fractie: public Pereche

{

private:

// Private functions

Fractie inversa();

public:

// Constructors & Destructor

Fractie();

Fractie(const Fractie&);

Fractie(double, double);

~Fractie();

// Functions

Fractie operator+(Fractie);

Fractie operator-(Fractie);

Fractie operator\*(Fractie);

Fractie operator/(Fractie);

Fractie operator^(int);

};

Cpp

#include "Fractie.h"

#include <iostream>

Fractie::Fractie(): Pereche()

{

}

Fractie::Fractie(double a, double b) : Fractie(a, b)

{

}

Fractie::~Fractie()

{

}

Fractie::Fractie(const Fractie &a)

{

firstOperand = a.firstOperand;

secondOperand = a.secondOperand;

}

Fractie Fractie::inversa()

{

Fractie rezultat;

rezultat.secondOperand = firstOperand;

rezultat.firstOperand = secondOperand;

return rezultat;

}

Fractie Fractie::operator+(Fractie A)

{

Fractie rezultat;

double aux;

if (secondOperand > A.secondOperand)

{

aux = secondOperand / A.secondOperand;

rezultat.secondOperand = A.secondOperand \* aux;

}

else if (secondOperand < A.secondOperand)

{

aux = A.secondOperand / secondOperand;

rezultat.secondOperand = secondOperand \* aux;

}

rezultat.firstOperand = firstOperand + A.firstOperand;

return rezultat;

}

Fractie Fractie::operator-(Fractie A)

{

Fractie rezultat;

double aux;

if (secondOperand > A.secondOperand)

{

aux = secondOperand / A.secondOperand;

rezultat.secondOperand = A.secondOperand \* aux;

}

else if (secondOperand < A.secondOperand)

{

aux = A.secondOperand / secondOperand;

rezultat.secondOperand = secondOperand \* aux;

}

rezultat.firstOperand = firstOperand - A.firstOperand;

return rezultat;

}

Fractie Fractie::operator\*(Fractie A)

{

Fractie rezultat;

rezultat.firstOperand = firstOperand \* A.firstOperand;

rezultat.secondOperand = secondOperand \* A.secondOperand;

return rezultat;

}

Fractie Fractie::operator/(Fractie A)

{

Fractie rezultat;

rezultat.firstOperand = firstOperand;

rezultat.secondOperand = secondOperand;

rezultat = rezultat \* A.inversa();

return rezultat;

}

Fractie Fractie::operator^(int power)

{

Fractie rezultat;

rezultat.secondOperand = pow(secondOperand, power);

rezultat.firstOperand = pow(firstOperand, power);

return rezultat;

}