Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет информационных технологий и программирования

Лабораторная работа № 4

Виртуальные функции

Вариант 6

Выполнил студент группы № M3111
Гонтарь Тимур Сергеевич
Подпись:

Санкт-Петербург
2023

Условие ЛР:

Реализовать все указанные интерфейсы (абстрактные базовые классы) для классов (согласно варианту):
A. Круг
B. Отрезок
C. Равносторонний треугольник
D. Прямоугольник
E. Шестиугольник
F. Параллелограмм
G. Равнобедренная трапеция
H. Эллипс (периметр можно считать по любой приближенной формуле: см. интернет, справочники и т.п.).

geomclasses.h

```
#ifndef LAB4_GEOMCLASSES_H
#define LAB4_GEOMCLASSES_H

// Интерфейс "Геометрическая фигура".
class IGeoFig {
public:
    virtual double square() = 0;

    virtual double perimeter() = 0;
};


// Вектор
class CVector2D {
public:
    double x, y;
};


// Интерфейс "Физический объект".
class IPhysObject {
public:
    virtual double mass() = 0;

    virtual CVector2D position() = 0;

    virtual bool operator==(const IPhysObject &ob) const = 0;

    virtual bool operator<(const IPhysObject &ob) const = 0;
};


// Интерфейс "Отображаемый"
class IPrintable {
public:
    virtual void draw() = 0;
};


// Интерфейс для классов, которые можно задать через диалог с пользователем.
class IDialogInitiable {
    virtual void initFromDialog() = 0;
};
```

```
// Интерфейс "Класс"
class BaseCObject {
public:
    virtual const char *classname() = 0;

    virtual unsigned int size() = 0;
};

class IFigure: public IGeoFig, public IPhysObject, public IPrintable, public
IDialogInitiable, public BaseCObject {};


#endif //LAB4_GEOMCLASSES_H
```

Parallelogram.h

```
#ifndef LAB4_Parallelogram_H
#define LAB4_Parallelogram_H

#include "../geomclasses.h"

class Parallelogram: public IFigure{
private:
    CVector2D start{};
    CVector2D a{};
    CVector2D b{};
public:
    Parallelogram();

    Parallelogram(CVector2D &, CVector2D &, CVector2D &);

    double sidelength(CVector2D &);

    double square() override;

    double perimeter() override;

    double mass() override;

    CVector2D position() override;

    bool operator==(const IPhysObject &) const override;

    bool operator<(const IPhysObject &) const override;

    void draw() override;

    void initFromDialog() override;

    const char * classname() override;

    unsigned int size() override;
};

#endif //LAB4_Parallelogram_H
```

Parallelolgram.cpp

```cpp
#include "parallelogram.h"

#include <iostream>
#include <cmath>

using std::cin;
using std::cout;
using std::endl;

//default constructor
Parallelogram::Parallelogram() {
    start = CVector2D{0, 0};
    a = CVector2D{0, 0};
    b = CVector2D{0, 0};
}

//constuctor with given vectors
Parallelogram::Parallelogram(CVector2D &s, CVector2D &one, CVector2D &two) {
    start = s;
    a = one;
    b = two;
}

//length of a side
double Parallelogram::sidelength(CVector2D &vec) {
    double len = sqrt(pow(vec.x, 2) + pow(vec.y, 2));
    return len;
}

//parallelogram square
double Parallelogram::square() {
    double ans = a.x * b.y - a.y * b.x;
    return ans;
}

//parallelogram perimeter
double Parallelogram::perimeter() {
    double ans = 2 * (sidelength(a) + sidelength(b));
    return ans;
}

//parallelogram mass
double Parallelogram::mass() {
    return square();
}

//parallelogram mass centre - diagonals intersection
CVector2D Parallelogram::position() {
    CVector2D *vec = new CVector2D;
    vec->x = (a.x + b.x) / 2 + start.x;
    vec->y = (a.y + b.y) / 2 + start.y;

    return *vec;
}

//mass comparing operator overloading
bool Parallelogram::operator==(const IPhysObject &other) const {
    double first = const_cast<Parallelogram &>(*this).mass();
    double second = const_cast<Parallelogram &>(dynamic_cast<const
Parallelogram &>(other)).mass();

    if (first == second) {
        return true;
    } else {
```

```
        return false;
    }
}

//mass comparing operator overloading
bool Parallelogram::operator<(const IPhysObject &other) const {
    double first = const_cast<Parallelogram &>(*this).mass();
    double second = const_cast<Parallelogram &>(dynamic_cast<const
Parallelogram &>(other)).mass();

    if (first < second) {
        return true;
    } else {
        return false;
    }
}

//draw a parallelogram
void Parallelogram::draw() {
    cout << "Пареллелограмм: " << classname() << endl;
    cout << "Координаты векторов параллелограмма:" << endl;
    cout << "Вектор координат стартовой точки: " << start.x << " " << start.y
<< endl;
    cout << "Вектор 1 стороны: " << a.x << " " << a.y << endl;
    cout << "Вектор 2 стороны: " << b.x << " " << b.y << endl;
    cout << endl;
}

//initialise a parallelogram from console
void Parallelogram::initFromDialog() {
    double sx, sy, ax, ay, bx, by;
    cout << "Введите координаты векторов для параллелограмма" << endl;
    cout << "Координаты вектора смещения для стартовой точки:" << endl;
    cin >> sx >> sy;
    cout << "Координаты вектора стороны 1:" << endl;
    cin >> ax >> ay;
    cout << "Координаты вектора стороны 2:" << endl;
    cin >> bx >> by;
    cout << endl;

    start = CVector2D{sx, sy};
    a = CVector2D{ax, ay};
    b = CVector2D{bx, by};
}

//parallelogram's classname
const char * Parallelogram::classname() {
    return typeid(*this).name();
}

//parallelogram's class size
unsigned int Parallelogram::size() {
    return sizeof(*this);
}
```

Ellipse.h

```
#ifndef LAB4_ELLIPSE_H
#define LAB4_ELLIPSE_H

#include "../geomclasses.h"
```

```
class Ellipse: public IFigure {
private:
    CVector2D center{};
    double big_axis;
    double small_axis;
public:
    Ellipse();

    Ellipse(CVector2D &, double, double);

    double square() override;

    double perimeter() override;

    double mass() override;

    CVector2D position() override;

    bool operator==(const IPhysObject &) const override;

    bool operator<(const IPhysObject &) const override;

    void draw() override;

    void initFromDialog() override;

    const char * classname() override;

    unsigned int size() override;
};

#endif //LAB4_ELLIPSE_H
```

Ellipse.cpp

```
#define pi M_PI

#include "ellipse.h"

#include <iostream>
#include <cmath>

using std::cin;
using std::cout;
using std::endl;

//default constructor
Ellipse::Ellipse() {
    center = CVector2D{0, 0};
    big_axis = 0;
    small_axis = 0;
}

//ellipse with given params
Ellipse::Ellipse(CVector2D &c, double small, double big) {
    center = c;
    big_axis = big;
    small_axis = small;
}

//ellipse square
double Ellipse::square() {
```

```
    double ans = pi * big_axis * small_axis;
    return ans;
}

//ellipse perimeter
double Ellipse::perimeter() {
    double ans = 4 * ((pi * big_axis * small_axis + (big_axis - small_axis))
/ (big_axis + small_axis));
    return ans;
}

//ellipse mass
double Ellipse::mass() {
    return square();
}

//ellipse mass centre - diagonals intersection
CVector2D Ellipse::position() {
    return center;
}

//mass comparing operator overloading
bool Ellipse::operator==(const IPhysObject &other) const {
    double first = const_cast<Ellipse &>(*this).mass();
    double second = const_cast<Ellipse &>(dynamic_cast<const Ellipse
&>(other)).mass();

    if (first == second) {
        return true;
    } else {
        return false;
    }
}

//mass comparing operator overloading
bool Ellipse::operator<(const IPhysObject &other) const {
    double first = const_cast<Ellipse &>(*this).mass();
    double second = const_cast<Ellipse &>(dynamic_cast<const Ellipse
&>(other)).mass();

    if (first < second) {
        return true;
    } else {
        return false;
    }
}

//draw an ellipse
void Ellipse::draw() {
    cout << "Эллипс: " << classname() << endl;
    cout << "Координаты центра эллипса:" << endl;
    cout << center.x << " " << center.y << endl;
    cout << "Большая полуось: " << big_axis << endl;
    cout << "Малая полуось: " << small_axis << endl;
    cout << endl;
}

//initialise an ellipse from console
void Ellipse::initFromDialog() {
    double cx, cy, big, small;
    cout << "Введите координаты центра эллипса: " << endl;
    cin >> cx >> cy;
    cout << "Введите большую и малую полуось: " << endl;
    cin >> big >> small;
```

```
    center = CVector2D{cx, cy};
    big_axis = big;
    small_axis = small;
}

//ellipse's classname
const char * Ellipse::classname() {
    return typeid(*this).name();
}

//ellipse's class size
unsigned int Ellipse::size() {
    return sizeof(*this);
}
```

Figuresset.h

```
#ifndef LAB4_FIGURESSET_H
#define LAB4_FIGURESSET_H

#include "ellipse/ellipse.h"
#include "parallelogram/parallelogram.h"

#include <set>

using std::set;

class Figures {
private:
    set<IFigure *> figs;
public:
    void addfigure();

    void display();

    double squareall();

    double perimeterall();

    CVector2D positionall();

    unsigned int memoryall();

};


#endif //LAB4_FIGURESSET_H
```

Figuresset.cpp

```
#include "ellipse/ellipse.h"
#include "parallelogram/parallelogram.h"
#include "figuresset.h"

#include "iostream"

using std::cin;
```

```
using std::cout;
using std::endl;

//add a figure to set
void Figures::addfigure() {
    int choice;
    cout << "Какую фигуру вы хотите добавить? Параллелограмм - 1, Эллипс -
2:" << endl;
    cin >> choice;
    while (choice != 1 and choice != 2) {
        cout << "Введите 1 или 2" << endl;
        cin >> choice;
    }

    if (choice == 1) {
        Parallelogram *temp = new Parallelogram;
        temp->initFromDialog();
        figs.insert(temp);
    } else {
        Ellipse *temp = new Ellipse;
        temp ->initFromDialog();
        figs.insert(temp);
    }
}

//display all figures
void Figures::display() {
    for (auto i:figs) {
        i->draw();
    }

    for (auto i:figs) {
        i->draw();
    }
}

//sum of all squares
double Figures::squareall() {
    double counter = 0;
    for (auto i:figs) {
        counter += i->square();
    }

    for (auto i:figs) {
        counter += i->square();
    }

    return counter;
}

//sum of all perimeters
double Figures::perimeterall() {
    double counter = 0;
    for (auto i:figs) {
        counter += i->perimeter();
    }

    for (auto i:figs) {
        counter += i->perimeter();
    }

    return counter;
}
```

```cpp
//central position of all
CVector2D Figures::positionall() {
    CVector2D ans{0, 0};
    double curx = 0;
    double cury = 0;
    double m = 0;

    for (auto i:figs) {
        curx += i->position().x * i->mass();
        cury += i->position().y * i->mass();
        m += i->mass();
    }

    for (auto i:figs) {
        curx += i->position().x * i->mass();
        cury += i->position().y * i->mass();
        m += i->mass();
    }

    ans.x = curx/m;
    ans.y = cury/m;

    return ans;
}

//memory of all classes
unsigned int Figures::memoryall() {
    unsigned int ans = 0;

    for (auto i:figs) {
        ans += i->size();
    }

    for (auto i:figs) {
        ans += i->size();
    }

    return ans;
}
```

Main.cpp

```cpp
#include "figuresset.h"
#include "parallelogram/parallelogram.h"
#include "ellipse/ellipse.h"

#include <iostream>
#include <string>
#include <windows.h>

using std::cout;
using std::cin;
using std::endl;
using std::string;

int main() {
    SetConsoleOutputCP(CP_UTF8);

    Figures figs1;

    cout << "1. Добавить фигуру в множество" << endl;
    cout << "2. Отобразить все фигуры" << endl;
    cout << "3. Суммарная площадь фигур" << endl;
```

```
    cout << "4. Суммарный периметр фигур" << endl;
    cout << "5. Центр масс всех фигур" << endl;
    cout << "6. Память на все экземляры классов" << endl;
    cout << "Символ точки - остановить программу" << endl;

    while (true) {
        string str;
        cin >> str;
        if (str == ".") {
            break;
        } else if (str == "1") {
            figs1.addfigure();
            cout << "Успех" << endl;
        } else if (str == "2") {
            figs1.display();
        } else if (str == "3") {
            cout << "Суммарная площадь равна " << figs1.squareall() << endl;
        } else if (str == "4") {
            cout << "Суммарный периметр равен " << figs1.perimeterall() <<
endl;
        } else if (str == "5") {
            cout << "Центр масс координаты " << figs1.positionall().x << " "
<< figs1.positionall().y << endl;
        } else if (str == "6") {
            cout << "Всего фигуры занимают " << figs1.memoryall() << " байт
памяти" << endl;
        }
    }


    cout << "До свидания" << endl;

    return 0;
}
```

Решение:

Вывод: в ходе данной лабораторной работы были реализованы интерфейсы, с помощью наследования и переопределения виртуальной функции.