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

Выполнил студент группы № M3111
Гаврилов Алексей Евгеньевич
Подпись:

Проверил:
Повышев Владислав Вячеславович

Санкт-Петербург
2022

Текст задания

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.).*

Функционал системы:

- Хранение множества фигур
- *Динамическое добавление фигур пользователем.* (через консоль)
- *Отобразить все фигуры.*
- *Суммарная площадь всех фигур.*
- *Суммарный периметр всех фигур.*
- *Центр масс всей системы.*
- *Память, занимаемая всеми экземплярами классов.*
- *Сортировка фигур между собой по массе.*

Вопросы для обдумывания:

- *Есть ли необходимость делать методы сравнения по массе виртуальными?*
- *Получится ли также перегрузить операторы сравнения для интерфейса BaseCObject чтобы сравнивать объекты по объему занимаемой памяти?*
- *Предположите, что в дальнейшем придется изменить код таким образом, чтобы фигуры (оставаясь сами по себе плоскими) задавались уже не в двумерном, а в трехмерном пространстве. Укажите как бы вы действовали? Что пришлось бы изменить?*

Решение с комментариями

//fun.cpp

```
#include <iostream>
#define _USE_MATH_DEFINES // для C++
#include <cmath>
#include <math.h>
#include <vector>
#include <string>
#include "fun.h"

using namespace std;

namespace fun{
        rect::rect(){
            this->rect::initFromDialog();
        }
        void rect::initFromDialog(){
            cout << endl;
            cout << "Print geo_fig's name: " << endl;
            cin >> name;
            cout << "Print point's cord a(x, y): " << endl;
            cin >> a.x >> a.y;
            cout << "Print point's cord b(x, y): " << endl;
            cin >> b.x >> b.y;
            cout << "Print point's cord c(x, y): " << endl;
            cin >> c.x >> c.y;
            cout << "Print point's cord d(x, y): " << endl;
            cin >> d.x >> d.y;
            cout << "Print massa of figure: " << endl;
            cin >> massa;
        }

        string rect::classname(){
            return name;
        }

        double rect::perimeter(){
            double temp = 0;

            double A = sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
            double B = sqrt((b.x-c.x)*(b.x-c.x)+(b.y-c.y)*(b.y-c.y));
            double C = sqrt((c.x-d.x)*(c.x-d.x)+(c.y-d.y)*(c.y-d.y));
            double D = sqrt((d.x-a.x)*(d.x-a.x)+(d.y-a.y)*(d.y-a.y));

            temp = A + B + C + D;

            return temp;
        }

        double rect::square(){
            double temp = 0;

            double d1 = sqrt((a.x-c.x)*(a.x-c.x)+(a.y-c.y)*(a.y-c.y));
            double d2 = sqrt((b.x-d.x)*(b.x-d.x)+(b.y-d.y)*(b.y-d.y));
            Vector2D V1, V2;

            V1.x = (a.x-c.x);
            V1.y = (a.y-c.y);

            V2.x = (b.x-d.x);
            V2.y = (b.y-d.y);

            double sin = sqrt(1 -
```

```
(abs(V1.x*V2.x+V1.y*V2.y)/(sqrt(V1.x*V1.x+V1.y*V1.y)*sqrt(V2.x*V2.x+V2.y*V2.y
)))*(abs(V1.x*V2.x+V1.y*V2.y)/(sqrt(V1.x*V1.x+V1.y*V1.y)*sqrt(V2.x*V2.x+V2.y*
V2.y))));

            temp = d1*d2*sin;

            return temp;
        }
        Vector2D rect::position(){
        Vector2D o;
        o.x = (a.x+c.x)/2;
        o.y = (a.y+c.y)/2;
        return o;
        }
        unsigned int rect::size(){
            unsigned int size_temp = sizeof(*this);
            return size_temp;
        }
        double rect::mass(){
            return massa;
        }
        void rect::draw(){
            cout << endl;
            cout << "Name: " << classname() << endl;
            cout << "Square: " << square() << endl;
            cout << "Perimeter: " << perimeter() << endl;
            cout << "Midel (x,y): " <<"(" << position().x <<"," <<
position().y << ")" << endl;
            cout << "Mass: " << mass() << endl;
            cout << "Memory: " << size() << endl;
        }

        bool rect::operator < (IPhysObject& other){
            if (other.mass() >= mass())
                return true;
            else
                return false;
        }

        bool rect::operator == (IPhysObject& other){
            if (other.mass() == mass())
                return true;
            else
                return false;
        }

        ellips::ellips(){
            this->ellips::initFromDialog();
        }

        void ellips::initFromDialog(){
            cout << endl;
            cout << "Print geo_fig's name: " << endl;
            cin >> name;
            cout << "Print point's cord a1(x, y): " << endl;
            cin >> a1.x >> a1.y;
            cout << "Print point's cord a2(x, y): " << endl;
            cin >> a2.x >> a2.y;
            cout << "Print point's cord b1(x, y): " << endl;
            cin >> b1.x >> b1.y;
            cout << "Print point's cord b2(x, y): " << endl;
            cin >> b2.x >> b2.y;
            cout << "Print massa of figure: " << endl;
            cin >> massa;
```

```
        }

        string ellips::classname(){
            return name;
        }


        double ellips::perimeter(){
            double temp = 0;

            double A = sqrt((a1.x-a2.x)*(a1.x-a2.x)+(a1.y-a2.y)*(a1.y-a2.y));
            double B = sqrt((b1.x-b2.x)*(b1.x-b2.x)+(b1.y-b2.y)*(b1.y-b2.y));

            temp = 2*M_PI*sqrt((A*A+B*B)/8);
            return temp;
        }

        double ellips::square(){
            double temp = 0;

            Vector2D o;
            o.x = (a1.x+a2.x)/2;
            o.y = (b1.x+b2.x)/2;
            double a = sqrt((a1.x-o.x)*(a1.x-o.x)+(a1.y-o.y)*(a1.y-o.y));
            double b = sqrt((b1.x-o.x)*(b1.x-o.x)+(b1.y-o.y)*(b1.y-o.y));

            temp = M_PI*a*b;

            return temp;
        }

        Vector2D ellips::position(){
            Vector2D o;
            o.x = (a1.x+a2.x)/2;
            o.y = (b1.x+b2.x)/2;
            return o;
        }
        double ellips::mass(){
            return massa;
        }
        unsigned int ellips::size()
        {
            unsigned int size_temp = sizeof(*this);
            return size_temp;
        }
        void ellips::draw()
        {
            cout << endl;
            cout << "Name: " << classname() << endl;
            cout << "Square: " << square() << endl;
            cout << "Perimeter: " << perimeter() << endl;
            cout << "Midel (x,y): " <<"(" << position().x <<"," <<
position().y << ")" << endl;
            cout << "Mass: " << mass() << endl;
            cout << "Memory: " << size() << endl;
        }
        bool ellips::operator < (IPhysObject& other){
            if (other.mass() >= mass())
                return true;
            else
                return false;
        }

        bool ellips::operator == (IPhysObject &other){
```

```
    if (other.mass() == mass())
        return true;
    else
        return false;
}

Menu::Menu(){}
Menu::Menu(int i){
    size = i;
    set.resize(size);
}

bool Menu::is_full(){
    if (temp_size < size)
        return true;
    else
        return false;
}

void Menu::show(){
    if (temp_size == 0)
        cout << endl << " Is null" << endl;
    else
    {
        cout << endl;
        for (int i = 0; i < temp_size; i++)
            set[i]->draw();
    }
}

void Menu::show_per(){
    double temp;
    if (temp_size == 0)
        cout << endl << " Is null" << endl;
    else
    {
        cout << endl;
        for (int i = 0; i < temp_size; i++)
            temp +=set[i]->perimeter();
        cout << endl << "Full per: " << temp << endl;
    }
}

void Menu::show_sqr(){
    double temp;
    if (temp_size == 0)
        cout << endl << " Is null" << endl;
    else
    {
        cout << endl;
        for (int i = 0; i < temp_size; i++)
            temp +=set[i]->square();
        cout << endl << "Full per: " << temp << endl;
    }
}

void Menu::show_mem(){
    unsigned int temp;
    if (temp_size == 0)
        cout << endl << " Is null" << endl;
    else
    {
        cout << endl;
        for (int i = 0; i < temp_size; i++)
```

```
                    temp +=set[i]->size();
                cout << endl << "Full per: " << temp << endl;
            }
        }

    void Menu::sort(){
        for (int i = 0; i < temp_size; i++) {
            for (int j = temp_size - 1; j > i; j--) {
                if (set[j - 1] < set[j]) {
                    figures *temp;
                    temp = set[j - 1];
                    set[j - 1] = set[j];
                    set[j] = temp;
                }
            }

        }
    }

    void Menu::positio() {
            Vector2D temp_midel;
            temp_midel.x=0;
            temp_midel.y=0;
            for (int i = 0; i < temp_size; i++){
               temp_midel.x +=set[i]->position().x;
               temp_midel.y +=set[i]->position().y;
            }
        cout << "Midel (x,y): " <<"(" << temp_midel.x <<"," << temp_midel.y
<< ")" << endl;
        }

        void Menu::start(){
            char str = '0';
            while (true){
                cin >> str;
                switch (str) {
                    case '+':
                    {
                        add<rect>();
                        break;
                    }
                    case '@':
                    {
                        add<ellips>();
                        break;
                    }
                    case '?':
                    {
                        show();
                        break;
                    }
                    case 'm':
                    {
                        show_mem();
                        break;
                    }
                    case 'p':
                    {
                        show_per();
                        break;
                    }
                    case 's':
                    {
                        show_sqr();
```

```
                        break;
                    }
                    case '/':
                    {
                        sort();
                        break;
                    }
                    case '&':
                    {
                        positio();
                        break;
                    }
                }
            }
        }

    };
```

```
//fun.h
// Created by Volirvag on 09.04.2022.
//

#ifndef LAB_4_FUN_H
#define LAB_4_FUN_H
#include <string>
using namespace std;
namespace fun{
    class IGeoFig{
    public:
        // Периметр.
        virtual double perimeter() = 0 ;
// Площадь.
        virtual double square() = 0 ;
    };

// Интерфейс "Отображаемый"
    class IPrintable{
    public:
// Отобразить на экране
// (выводить в текстовом виде параметры фигуры).
        virtual void draw() = 0 ;
    };
    // Интерфейс "Класс"
    class BaseCObject{
    public:
// Имя класса (типа данных).
        virtual string classname() = 0;
// Размер занимаемой памяти.
        virtual unsigned int size() = 0;
    };

    class Vector2D{
    public :
        double x, y;
    };
    class IPhysObject{
    public :
// Масса, кг.
        virtual double mass() = 0;
// Координаты центра масс, м.
        virtual Vector2D position() = 0 ;
        // Сравнение по массе.
```

```
        virtual bool operator == (IPhysObject &ob1) = 0;
// Сравнение по массе.
        virtual bool operator < (IPhysObject &ob2) = 0;
    };

// Вектор


// Интерфейс "Физический объект".




// Интерфейс для классов, которые можно задать через диалог с пользователем.
    class IDialogInitiable{
// Задать параметры объекта с помощью диалога с пользователем.
    virtual void initFromDialog () = 0 ;
    };
    class figures: public IGeoFig, public IPhysObject, public IPrintable,
public IDialogInitiable, public BaseCObject{};
    class rect: public figures{
    private:
        Vector2D a, b, c, d;
        string name = "Rectangle";
        double massa = 0;
    public:
        rect();

        void initFromDialog() override;

        string classname() override;

        unsigned int size() override;

        double perimeter() override;


        double square() override;

        Vector2D position() override;

        double mass() override;

        void draw() override;


        bool operator < (IPhysObject &other) override;

        bool operator == (IPhysObject &other) override;

    };

    class ellips: public figures{
    private:
        Vector2D a1, b1, a2, b2;
        string name = "Ellips";
        double massa = 0;
    public:
        ellips();

        void initFromDialog() override;


        string classname() override;
```

```
        unsigned int size() override;

        double perimeter() override;

        double square() override;

        Vector2D position() override;

        double mass() override;

        void draw() override;

        bool operator < (IPhysObject &other) override;

        bool operator == (IPhysObject &other) override;
    };
    class Menu{
    private:
        int size = 0;
        int temp_size = 0;
        vector<figures*> set;
    public:
        Menu();
        Menu(int i);

        bool is_full();

        template<class T>
        bool add(){
            if (is_full()){
                set[temp_size] = new T;
                temp_size++;
                return true;
            }
            else
                return false;
        };

        void show();

        void show_per();

        void show_sqr();

        void show_mem();

        void start();

        void sort();

        void positio();

    };



}
```

```
#endif //LAB_4_FUN_H
```

//main

```
#include <iostream>
#define _USE_MATH_DEFINES // для C++
#include <cmath>
#include <math.h>
#include <vector>
#include <string>
#include "fun.h"

using namespace std;

int main() {
    int size = 0;
    cin >> size;
    fun::Menu m(size);
    m.start();
    return 0;
}
```