Лабораторные работы по курсу

"Объектно-ориентированный анализ и проектирование".

Лабораторная работа №3

**Вариант 3**

Выполнили студенты  
гр. КТбо2-6

Бояринов В.В.

Ландин Ю.В.

Таганрог 2019

Оглавление

[Задание 1](#_Toc528784612)

[Используемые математические зависимости и алгоритмы 1](#_Toc528784613)

[Диаграмма классов 2](#_Toc528784614)

[Перечень сигнатур методов с комментариями 3](#_Toc528784615)

[Листинг: классы, методы, main 5](#_Toc528784616)

[Контрольные вопросы 14](#_Toc528784617)

# Задание

В лабораторной работе должна быть создана программа, создающая объекты двух классов (T1, T2) из семи производных классов и выполняющая некоторые операции над объектами этих классов. Базовым является класс Shape.

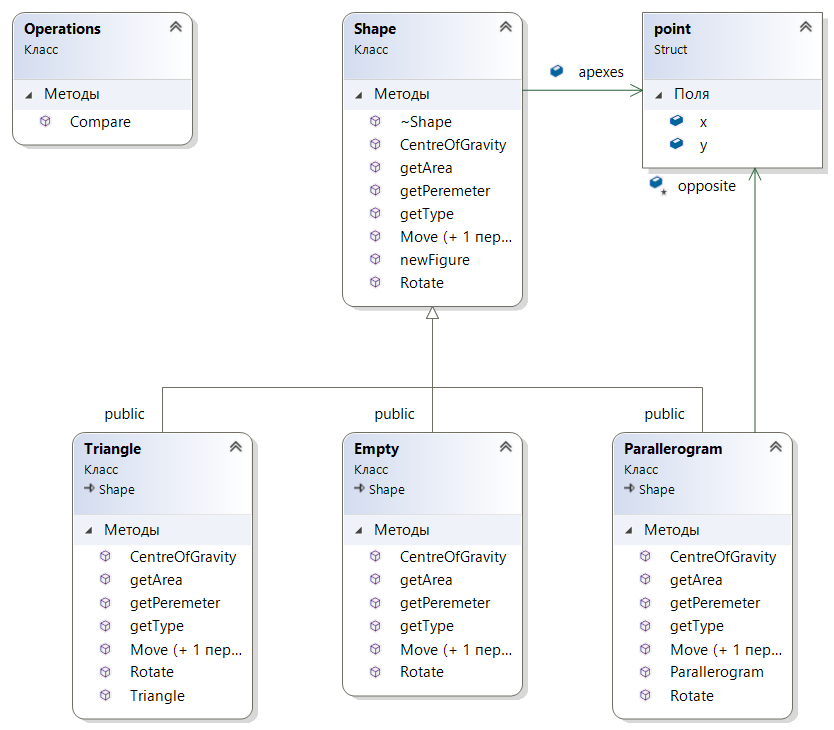
Производные классы должны создавать плоские объекты следующих типов: Треугольник, Параллелограмм. Для каждого типа фигуры вычисляются площадь, центр тяжести и другие атрибуты, а также должны быть предусмотрены виртуальные методы Вращения (Rotate) и Перемещения (Move).

Особенностью программы является использование статического метода createShape, который можно вызывать без создания объекта соответствующего класса.

# Используемые математические зависимости и алгоритмы

* Вычисление площади, периметра и центра тяжести. https://bit.ly/2CTavAf

# Диаграмма классов



# Перечень сигнатур методов с комментариями

* Методы абстрактного класса Shape
  + - virtual double getArea() // виртуальный метод, возвращающий площадь объекта
    - virtual double getPeremeter() // виртуальный метод, возвращающий периметр объекта
    - virtual string getType() // виртуальная функция, возвращающий тип объекта
    - static Shape\* newFigure(char ch); // статический метод, создающий объект одного из дочерних для "Shape" классов
    - virtual ~Shape() // дескриптор, очищающий выделеную под "apexes" память
    - virtual point CentreOfGravity() = 0; // виртуальный метод, возвращающий центр тяжести объекта
    - virtual void Rotate(int angle) = 0; // виртуальный метод, который поворачивает фигуру
    - virtual void Move(point offset) = 0; // виртуальный метод, который двигает фигуру
    - virtual void Move(int x, int y) = 0; // виртуальный метод, который двигает фигуру
* Методы класса Parallerogram
  + - Parallerogram(); // конструктор, инициализирующий фигуру
    - string getType(); // виртуальная функция, возвращающий тип объекта
    - void Rotate(int angle); // виртуальный метод, который поворачивает фигуру
    - void Move(point offset); // виртуальный метод, который двигает фигуру
    - void Move(int x, int y); // виртуальный метод, который двигает фигуру
    - double getArea(); // виртуальный метод, возвращающий площадь объекта
    - double getPeremeter(); // виртуальный метод, возвращающий периметр объекта
    - point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта
* Методы класса Triangle

Triangle(); // конструктор, инициализирующий фигуру

string getType(); // виртуальный метод, возвращающий тип объекта

void Rotate(int angle); // виртуальный метод, который вращает фигуру

void Move(point offset); // виртуальный метод, который двигает фигуру

void Move(int x, int y); // виртуальный метод, который двигает фигуру

double getArea(); // виртуальный метод, возвращающий площадь объекта

double getPeremeter(); // виртуальный метод, возвращающий периметр объекта

point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта

* Методы класса Empty

string getType(); // виртуальный метод, возвращающий тип объекта

void Rotate(int angle); // виртуальный метод, который вращает фигуру

void Move(point offset); // виртуальный метод, который двигает фигуру

void Move(int x, int y); // виртуальный метод, который двигает фигуру

double getArea(); // виртуальный метод, возвращающий площадь объекта

double getPeremeter(); // виртуальный метод, возвращающий периметр объекта

point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта

Листинг: классы, методы, main

//Shapes.h

#pragma once

#include "main.h"

class Shape // базовый класс для всех фигур

{

public:

point\* apexes; // точки, из которых строится фигура

virtual double getArea() // виртуальный метод, возвращающий площадь объекта

{

return 0;

}

virtual double getPeremeter() // виртуальный метод, возвращающий периметр объекта

{

return 0;

}

virtual string getType() // виртуальная функция, возвращающий тип объекта

{

return "null";

}

static Shape\* newFigure(char ch); // статический метод, создающий объект одного из дочерних для "Shape" классов

virtual ~Shape() // дескриптор, очищающий выделеную под "apexes" память

{

delete[] apexes;

}

virtual point CentreOfGravity() = 0; // виртуальный метод, возвращающий центр тяжести объекта

virtual void Rotate(int angle) = 0; // виртуальный метод, который поворачивает фигуру

virtual void Move(point offset) = 0; // виртуальный метод, который двигает фигуру

virtual void Move(int x, int y) = 0; // виртуальный метод, который двигает фигуру

};

class Parallerogram : public Shape // класс "параллерограмм"

{

protected:

point opposite = { 0,0 }; // точка, находящаяся дальше всех от первой введёной

public:

Parallerogram(); // конструктор, инициализирующий фигуру

string getType(); // виртуальная функция, возвращающий тип объекта

void Rotate(int angle); // виртуальный метод, который поворачивает фигуру

void Move(point offset); // виртуальный метод, который двигает фигуру

void Move(int x, int y); // виртуальный метод, который двигает фигуру

double getArea(); // виртуальный метод, возвращающий площадь объекта

double getPeremeter(); // виртуальный метод, возвращающий периметр объекта

point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта

};

Parallerogram::Parallerogram()

{

apexes = new point[4];

for (int i = 0; i < 4; i++)

{

cout << "Введите координаты " << i + 1 << " вершины через пробел: "; cin >> apexes[i].x >> apexes[i].y;

}

double len[3];

double max\_len = 0;

int maxLenIndex = 0;

for (int i = 1; i < 4; i++)

{

len[i - 1] = FindLength(apexes[0], apexes[i]);

if (max\_len < len[i - 1])

{

max\_len = len[i - 1];

maxLenIndex = i;

}

}

opposite = apexes[maxLenIndex];

}

string Parallerogram::getType()

{

return "Parallerogram";

}

void Parallerogram::Rotate(int angle)

{

for (int i = 0; i < 4; i++)

{

point tmp;

tmp.x = apexes[i].x;

tmp.y = apexes[i].y;

apexes[i].x = tmp.x \* cos(angle) - tmp.y \* sin(angle);

apexes[i].y = tmp.x \* sin(angle) + tmp.y \* cos(angle);

}

}

void Parallerogram::Move(point offset)

{

for (int i = 0; i < 4; i++)

apexes[i] = apexes[i] + offset;

}

void Parallerogram::Move(int x, int y)

{

for (int i = 0; i < 4; i++)

{

apexes[i].x = apexes[i].x + x;

apexes[i].y = apexes[i].y + y;

}

}

double Parallerogram::getArea()

{

double AB = sqrt(pow(apexes[1].x - apexes[0].x, 2) + pow(apexes[1].y - apexes[0].y, 2));

double AD = sqrt(pow(apexes[3].x - apexes[0].x, 2) + pow(apexes[3].y - apexes[0].y, 2));

double cosA = (((apexes[1].x - apexes[0].x) \* (apexes[3].x - apexes[0].x)) - ((apexes[1].y - apexes[0].y) \* (apexes[3].y - apexes[0].y))) / (AB \* AD);

double sinA = sqrt(1 - pow(cosA, 2));

double sum = AB \* AD \* sinA;

return sum;

}

double Parallerogram::getPeremeter()

{

double sum = 0;

for (int i = 0; i < 4; i++)

{

sum += FindLength(apexes[i], apexes[(i + 1) % 4]);

}

return sum;

}

point Parallerogram::CentreOfGravity()

{

point centre;

centre.x = (apexes[0].x + opposite.x) / 2;

centre.y = (apexes[0].y + opposite.y) / 2;

return centre;

}

class Triangle : public Shape // класс "треугольник"

{

public:

Triangle(); // конструктор, инициализирующий фигуру

string getType(); // виртуальный метод, возвращающий тип объекта

void Rotate(int angle); // виртуальный метод, который вращает фигуру

void Move(point offset); // виртуальный метод, который двигает фигуру

void Move(int x, int y); // виртуальный метод, который двигает фигуру

double getArea(); // виртуальный метод, возвращающий площадь объекта

double getPeremeter(); // виртуальный метод, возвращающий периметр объекта

point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта

};

Triangle::Triangle()

{

apexes = new point[3];

for (int i = 0; i < 3; i++)

{

cout << "Введите координаты " << i + 1 << " вершины через пробел:"; cin >> apexes[i].x >> apexes[i].y;

}

}

string Triangle::getType()

{

return "Triangle";

}

void Triangle::Rotate(int angle)

{

for (int i = 0; i < 3; i++)

{

point tmp;

tmp.x = apexes[i].x;

tmp.y = apexes[i].y;

apexes[i].x = tmp.x \* cos(angle) - tmp.y \* sin(angle);

apexes[i].y = tmp.x \* sin(angle) + tmp.y \* cos(angle);

}

}

void Triangle::Move(point offset)

{

for (int i = 0; i < 3; i++)

apexes[i] = apexes[i] + offset;

}

void Triangle::Move(int x, int y)

{

for (int i = 0; i < 3; i++)

{

apexes[i].x = apexes[i].x + x;

apexes[i].y = apexes[i].y + y;

}

}

double Triangle::getArea()

{

double sum = 0.5 \* (((apexes[0].x - apexes[2].x) \* (apexes[1].y - apexes[2].y)) - ((apexes[0].y - apexes[2].y) \* (apexes[1].x - apexes[2].x)));

if (sum < 0)

sum \*= -1;

return sum;

}

double Triangle::getPeremeter()

{

double sum = 0;

for (int i = 0; i < 3; i++)

{

sum += FindLength(apexes[i], apexes[(i + 1) % 3]);

}

return sum;

}

point Triangle::CentreOfGravity()

{

point centre;

centre.x = (apexes[0].x + apexes[1].x + apexes[2].x) / 3.0;

centre.y = (apexes[0].y + apexes[1].y + apexes[2].y) / 3.0;

return centre;

}

class Empty : public Shape // пустой класс, создающийся без если требуется создать фигуру, не описанную в программе заранее

{

public:

string getType(); // виртуальный метод, возвращающий тип объекта

void Rotate(int angle); // виртуальный метод, который вращает фигуру

void Move(point offset); // виртуальный метод, который двигает фигуру

void Move(int x, int y); // виртуальный метод, который двигает фигуру

double getArea(); // виртуальный метод, возвращающий площадь объекта

double getPeremeter(); // виртуальный метод, возвращающий периметр объекта

point CentreOfGravity(); // виртуальный метод, возвращающий центр тяжести объекта

};

string Empty::getType()

{

return "Invalid Shape";

}

void Empty::Rotate(int angle)

{

cout << "Can't rotate: Invalid Shape" << endl;

}

void Empty::Move(point offset)

{

cout << "Can't rotate: Invalid Shape" << endl;

}

void Empty::Move(int x, int y)

{

cout << "Can't move: Invalid Shape" << endl;

}

double Empty::getArea()

{

return 0;

}

double Empty::getPeremeter()

{

return 0;

}

point Empty::CentreOfGravity()

{

return point{ 0, 0 };

}

Shape\* Shape::newFigure(char ch)

{

ch = toupper(ch);

Shape\* newClass = 0;

switch (ch)

{

case 'T':

newClass = new Triangle();

break;

case 'P':

newClass = new Parallerogram();

break;

default:

newClass = new Empty();

break;

}

return newClass;

}

// Operations.h  
#pragma once

#include "main.h"

#include "Shapes.h"

static class Operations // статический класс с операциями над объектами

{

public:

static bool Compare(Shape\* obj1, Shape\* obj2) // сравнение площадей двух объектов

{

return obj1->getArea() > obj2->getArea();

}

};  
  
// main.h  
#pragma once

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

struct point // структура "точка"

{

double x, y;

};

inline double FindLength(point a, point b) // вычисление длины между двумя точками

{

return sqrt(pow((a.x - b.x), 2) + pow((a.y - b.y), 2));

}

ostream& operator<<(ostream& os, const point& pt) // перегрузка оператора "<<" для удобного вывода точек

{

os << "X: " << pt.x << "|Y: " << pt.y;

return os;

}

point operator + (const point &p1, const point &p2) // перегрузка оператора "+" для сложения точек

{

point response;

response.x = p1.x + p2.x;

response.y = p1.y + p2.y;

return response;

}

// main.cpp

#include <iostream>

#include "Shapes.h"

#include "Operations.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

// инициализация трёх переменных под объекты

Shape\* o1 = NULL;

Shape\* o2 = NULL;

Shape\* o3 = NULL;

// создание треугольника и вывод его типа на экран

o1 = Shape::newFigure('t');

cout << o1->getType() << endl;

// создание паралеррограмма и вывод его типа на экран

o2 = Shape::newFigure('p');

cout << o2->getType() << endl;

// создание объекта класса "Empty" и вывод его типа на экран

o3 = Shape::newFigure('2');

cout << o3->getType() << endl;

// сравнение двух объектов через метод Operations::Compare и вывод результата на экран

cout << o1->getType() << " больше " << o2->getType() << ": " << Operations::Compare(o1, o2) << endl;

return 0;

}

# 

# Контрольные вопросы

1. Различают 4 элемента определения функции: тип, имя, список параметров, тело. Какими элементами могут отличаться экземпляры одной виртуальной функции, находящиеся в разных производных классах?

2. В каких случаях применяется фабричный метод?

3. Если вызывается виртуальная функция производного класса, в котором она не определена, то в какую точку переходит управление программой?

4. При описании виртуальных функций встречаются выражения “динамическое связывание”, “статическое связывание”. О связывании чего идет речь?

5. Укажите на недостаток использования виртуальной функции.

6. Зачем нужен виртуальный деструктор? Продемонстрируйте его поведение на собственной программе.

7. Можно ли использовать для вызова виртуальных функций не указатели, а ссылки?

Автор Преподаватель

Бояринов В.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ландин Ю.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 29.03.2019