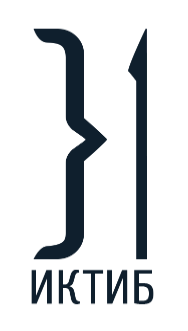
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ



**ЛАБОРАТОРНАЯ РАБОТА №3**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Динамический выбор типа объектов»**

Вариант №2

Выполнили:

Студенты группы

КТбо2-6

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Беликова Ю. А. |
|  | *подпись* |  |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Чепурная Т.В. |
|  | *подпись* |  |

Проверил:

ассистент кафедры

МОП ЭВМ

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Чичерина К. С. |
|  | *подпись* |  |

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

# Задание

Необходимо создать программу, создающую объекты двух классов (треугольник, прямоугольник). Эти классы должны быть производными от класса Shape.

# Спецификация классов

1. Базовый класс Shape

Содержит структуру Point (координаты).

struct Point

{

double x;

double y;

};

Методы:

virtual double Area() const = 0;- Определение площади фигуры

virtual Point CenterOfGravity() const = 0; - Определение центра тяжести

void Move(int x, int y);- Перемещение фигуры

void Rotate(const int angle);- Вращение фигуры

string ToString() const;- Возвращает координаты фигуры

virtual void TrueShape() const = 0;- Определяет есть ли фигура

virtual double GetRadius() const = 0;- Получение радиуса

double GetLength(const Point& dot1, const Point& dot2) const; - Получение длины стороны

Поля:

Point\* \_cords; - указатель на координаты

int \_countSides; - количество сторон

double\* \_lengthSide; - длина стороны

const double PI;

2.Производный класс Triangle

Конструктор:

Triangle(const Point\* cords) : Shape(cords, 3) { TrueShape(); }

Методы:

double Area() const override; - вычисление площади

Point CenterOfGravity() const override; - определение центра тяжести

void TrueShape() const override; - есть ли такая фигура

double GetRadius() const override; - получение радиуса

double Perimeter() const; - узнаем периметр

3.Производный класс Rectangle

Конструктор:

Rectangle(const Point\* cords) : Shape(cords, 4) { TrueShape(); }

Методы:

double Area() const override; - вычисление площади

Point CenterOfGravity() const override; - определение центра тяжести

void TrueShape() const override; - есть ли такая фигура

double GetRadius() const override; получение радиуса

4. Класс Factory

Static Shape\* createShape(const Point\* cords, int number); - фабричный метод

5. Класс Operation

int Compare(const Shape& shape1, const Shape& shape2) const; - сравнение площадей

bool IsIntersect(const Shape& shape1, const Shape& shape2) const; - пересечение фигур

bool IsInclude(const Shape& shape1, const Shape& shape2) const; - включение фигур

double GetLength(const Point& dot1, const Point& dot2) const; - получение длины

6. Класс ConsoleInteractor

Конструктор:

ConsoleInteractor();

Деструктор:

~ConsoleInteractor();

Методы:

void Init() const; - меню

private:

int GetIndex(const string& name) const; - проверяет есть ли введенное имя

void Help() const; - помощь

void CreateShape() const; - создание фигуры

void PrintShape() const; - печать координат фигуры

void Compare() const; - сравнение площадей фигур

void IsIntersect() const; - пересечение фигур

void IsInclude() const; - включение фигур

void Move() const; - перемещение фигур

void Rotate() const; - вращение фигур

void Area() const; - площадь фигуры

void CenterOfGravity() const; - центр тяжести

Поля:

Shape\*\* \_storage = nullptr; - массив указателей

string\* \_names = nullptr; - храним имена фигур

int \_count = 0; - количество фигур

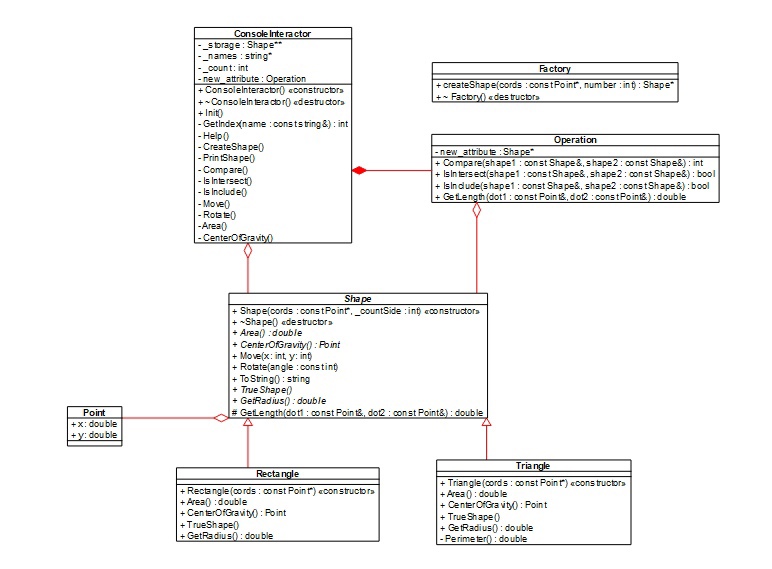
# Используемые математические зависимости и алгоритмы

Площадь треугольника находим по формуле: S =sqrt(p(p-a) (p-b)(p-c)), где p - полупериметр, a,b,c – стороны треугольника.

Площадь прямоугольника находим по формуле: S = a \* b, где, a,b – стороны прямоугольника.

Вычисление радиуса треугольника: R= a\*b\*c/ (4 \* sqrt(p\*(p-a)\*(p-b)\*(p-c))), где a,b,c – стороны треугольника, p – полупериметр.

# Диаграмма классов



# Листинг программы

Проект включает 13 файлов: заголовочные ­­– Shape.h, Function.h, ConsoleInteractor.h, Triangle.h, Rectangle.h, Operation.h, реализации: Shape.cpp, Function.cpp, ConsoleInteractor.cpp, Triangle.cpp, Rectangle.cpp, Operation.cpp и main.cpp.

Main.cpp

#include <iostream>

#include "ConsoleInteractor.h"

#include <vld.h>

int main()

{

ConsoleInteractor console;

console.Init();

return 0;

}

ConsoleInteractor.h

#pragma once

#include <iostream>

#include "Operation.h"

#include "Factory.h"

using std::string;

class ConsoleInteractor

{

public:

ConsoleInteractor();

~ConsoleInteractor();

void Init() const;

private:

int GetIndex(const string& name) const;

void Help() const;

void CreateShape() const;

void PrintShape() const;

void Compare() const;

void IsIntersect() const;

void IsInclude() const;

void Move() const;

void Rotate() const;

void Area() const;

void CenterOfGravity() const;

Shape\*\* \_storage = nullptr;// массив указателей на объекты класса shape

string\* \_names = nullptr;//указатель на имена фигур

int \_count = 0;

};

ConsoleInteractor.cpp

#include <iostream>

#include "ConsoleInteractor.h"

using std::cin;

using std::cout;

using std::cerr;

ConsoleInteractor::ConsoleInteractor()

{

cout << "how many shape?\n";

cin >> \_count;

\_names = new string[\_count];

\_storage = new Shape \* [\_count];// массив указателей на объекты класса shape

for (int i=0;i<\_count;i++)

CreateShape();

}

ConsoleInteractor::~ConsoleInteractor()

{

for (int i = 0; i < \_count; i++)

{

if (\_names[i] != "")

{

delete \_storage[i];

}

}

delete[] \_names;

delete[] \_storage;

}

void ConsoleInteractor::CreateShape() const

{

string name;

int countVertex;

cout << "name the Shape\n";

cin >> name;

cout << "number of Vertex?\n";

cin >> countVertex;

if (countVertex == 3 || countVertex == 4)

{

Point\* dots = new Point[countVertex];

cout << "Enter dots clockwise or counterclockwise\n";

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

{

cout << "\n" << "x: ";

cin >> dots[i].x;

cout << "y: ";

cin >> dots[i].y;

}

bool FreePlace = false;

int index = 0;

for (int i = 0; i < \_count && !FreePlace; i++)//выдается индекс для имени

{

if (\_names[i] == "")

{

FreePlace = true;

index = i;

}

}

if (FreePlace)

{

Shape\* shape = nullptr;

try

{

shape = Factory::createShape(dots, countVertex);

\_names[index] = name;

\_storage[index] = shape;

}

catch (const std::logic\_error& src)

{

cout << src.what() << "\n";

}

}

else

{

cout << "you run out of place, end the program (click on 11) or choose operations for shapes\n";

}

delete[] dots;

}

else

{

cout << "WRONG\_SHAPE\n";

}

}

int ConsoleInteractor::GetIndex(const string& name) const

{

for (int i = 0; i < \_count; i++)

{

if (name == \_names[i])

{

return i;

}

}

return -1;

}

void ConsoleInteractor::PrintShape() const

{

string name;

cout << "Name to print?\n";

cin >> name;

if (GetIndex(name) != -1)

{

cout << \_storage[GetIndex(name)]->ToString() << "\n";

}

else

{

cout << "Name is not found\n";

}

}

void ConsoleInteractor::Compare() const

{

Operation operation;

string name1, name2;

cout << "Name of first shape\n";

cin >> name1;

cout << "Name of second shape\n";

cin >> name2;

if ((GetIndex(name1) && GetIndex(name2)) != -1)

{

switch (operation.Compare(\*\_storage[GetIndex(name1)], \*\_storage[GetIndex(name2)]))

{

case 1:

cout << "First shape larger than second\n";

break;

case -1:

cout << "Second shape larger than first\n";

break;

case 0:

cout << "Shapes have the same area\n";

break;

}

}

else

{

cout << "Name in not found\n";

}

}

void ConsoleInteractor::Move() const

{

string name;

int x, y;

cout << "Name of shape?\n";

cin >> name;

if (GetIndex(name) != -1)

{

cout << "How much x and y\n";

cin >> x >> y;

\_storage[GetIndex(name)]->Move(x, y);

}

else

{

cout << "Name is not found\n";

}

}

void ConsoleInteractor::Rotate() const

{

string name;

int angle;

cout << "Name of shape\n";

cin >> name;

if (GetIndex(name) != -1)

{

cout << "Enter angle\n";

cin >> angle;

\_storage[GetIndex(name)]->Rotate(angle);

}

else

{

cout << "Name is not found\n";

}

}

void ConsoleInteractor::Area() const

{

string name;

cout << "Name of shape\n";

cin >> name;

if (GetIndex(name) != -1)

{

cout << "Area of this shape is " << \_storage[GetIndex(name)]->Area() << "\n";

}

else

{

cout << "Name is not found\n";

}

}

void ConsoleInteractor::IsIntersect() const

{

Operation operation;

string name1, name2;

cout << "Name of first shape\n";

cin >> name1;

cout << "Name of second shape\n";

cin >> name2;

if (GetIndex(name1) != -1 && (GetIndex(name2)) != -1)

{

if (operation.IsIntersect(\*\_storage[GetIndex(name1)], \*\_storage[GetIndex(name2)]))

{

cout << "Shapes intersect\n";

}

else

{

cout << "Shapes don't intersect\n";

}

}

else

{

cout << "Name in not found\n";

}

}

void ConsoleInteractor::CenterOfGravity() const

{

string name;

cout << "Name of shape\n";

cin >> name;

if (GetIndex(name) != -1)

{

Point centerOfGraviry = \_storage[GetIndex(name)]->CenterOfGravity();

cout << "Center of gravity this shape " << centerOfGraviry.x << " " << centerOfGraviry.y << "\n";

}

else

{

cout << "Name is not found\n";

}

}

void ConsoleInteractor::IsInclude() const

{

Operation operation;

string name1, name2;

cout << "Name of first shape\n";

cin >> name1;

cout << "Name of second shape\n";

cin >> name2;

if (GetIndex(name1) != -1 && (GetIndex(name2) != -1))

{

if (operation.IsInclude(\*\_storage[GetIndex(name1)], \*\_storage[GetIndex(name2)]))

{

cout << "Shapes is include\n";

}

else

{

cout << "Shapes don't include\n";

}

}

else

{

cout << "Name in not found\n";

}

}

void ConsoleInteractor::Help() const

{

cout << "1 - Create Shape if you entered wrongs dots of shape \n2 - Help\n3 - Compare\n4 - IsInclude \n5 - IsIntersect\n6 - Print Shape\n7 - Move\n8 - Rotate\n9 - Area\n10 - Get Center\n11 - Exit\n";

}

void ConsoleInteractor::Init() const

{

Help();

bool flag = true;

int number;

while (flag)

{

cout << "Enter the number\n";

cin >> number;

if (number == 1)

{

CreateShape();

}

else if (number == 2)

{

Help();

}

else if (number == 3)

{

Compare();

}

else if (number == 4)

{

IsInclude();

}

else if (number == 5)

{

IsIntersect();

}

else if (number == 6)

{

PrintShape();

}

else if (number == 7)

{

Move();

}

else if (number == 8)

{

Rotate();

}

else if (number == 9)

{

Area();

}

else if (number == 10)

{

CenterOfGravity();

}

else if (number == 11)

{

flag = false;

}

}

}

Operations.h

#pragma once

#include "Shape.h"

class Operation

{

public:

int Compare(const Shape& shape1, const Shape& shape2) const;

bool IsIntersect(const Shape& shape1, const Shape& shape2) const;

bool IsInclude(const Shape& shape1, const Shape& shape2) const;

private:

double GetLength(const Point& dot1, const Point& dot2) const;

};

Operations.cpp

#include "Operation.h"

int Operation::Compare(const Shape& shape1, const Shape& shape2) const

{

double area1 = shape1.Area();

double area2 = shape2.Area();

if (area1 > area2)

{

return 1;

}

else if (area1 < area2)

{

return -1;

}

return 0;

}

double Operation::GetLength(const Point& dot1, const Point& dot2) const

{

double length = sqrt(pow(dot2.x - dot1.x, 2) + pow(dot2.y - dot1.y, 2));

return length;

}

bool Operation::IsIntersect(const Shape& shape1, const Shape& shape2) const

{

double radiusF = shape1.GetRadius();

double radiusS = shape2.GetRadius();

Point centerF = shape1.CenterOfGravity();

Point centerS = shape2.CenterOfGravity();

double length = GetLength(centerF, centerS);

if (radiusF + radiusS >= length && length >= abs(radiusF - radiusS))

{

return true;

}

else

{

return false;

}

}

bool Operation::IsInclude(const Shape& shape1, const Shape& shape2) const

{

double radiusF = shape1.GetRadius();

double radiusS = shape2.GetRadius();

Point centerF = shape1.CenterOfGravity();

Point centerS = shape2.CenterOfGravity();

double length = GetLength(centerF, centerS);

if (length <= abs(radiusF - radiusS))

{

return true;

}

return false;

}

Factory.h

#pragma once

#include "Rectangle.h"

#include "Triangle.h"

class Factory

{

public:

static Shape\* createShape(const Point\* cords, int number);

};

Factory.cpp

#include "Factory.h"

#include <stdexcept>

using std::logic\_error;

//фабричный метод

Shape\* Factory::createShape(const Point\* cords, int number)

{

if (number == 3)

{

return new Triangle(cords);

}

else if (number == 4)

{

return new Rectangle(cords);

}

else

throw logic\_error("WRONG\_SHAPE");

}

Shape.h

#pragma once

#include <iostream>

#include <string>

#include <cmath>

using std::string;

struct Point

{

double x;

double y;

};

class Shape

{

public:

Shape(const Point\* cords, int \_countSide);

virtual ~Shape();

virtual double Area() const = 0;

virtual Point CenterOfGravity() const = 0;

void Move(int x, int y);

void Rotate(const int angle);

string ToString() const;

virtual void TrueShape() const = 0;

virtual double GetRadius() const = 0;

protected:

Point\* \_cords;

double GetLength(const Point& dot1, const Point& dot2) const;

int \_countSides;

double\* \_lengthSide;

const double PI;

};

Shape.cpp

#include "Shape.h"

Shape::~Shape()

{

delete[] \_cords;

delete[] \_lengthSide;

}

Shape::Shape(const Point\* cords, int countSide) : PI(acos(-1)), \_countSides(countSide) //определения числа Пи — с помощью одной из обратных тригонометрических функций с помощью арккосинуса

{

\_cords = new Point[\_countSides];

\_lengthSide = new double[\_countSides];

memcpy(\_cords, cords, sizeof(Point) \* \_countSides);

for (int i = 0; i < \_countSides; i++)

{

\_lengthSide[i] = GetLength(cords[i], cords[(i + 1) % \_countSides]);

}

}

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

{

for (int i = 0; i < \_countSides; i++)

{

\_cords[i].x += x;

\_cords[i].y += y;

}

}

double Shape::GetLength(const Point& dot1, const Point& dot2) const

{

double length = (sqrt(pow(dot2.x - dot1.x, 2) + pow(dot2.y - dot1.y, 2)));

return length;

}

string Shape::ToString() const

{

std::string res = "";

for (int i = 0; i < \_countSides; i++)

{

res += std::to\_string(i) + ") x: " + std::to\_string(\_cords[i].x) + " y: " + std::to\_string(\_cords[i].y) + "\n";

}

return res;

}

void Shape::Rotate(const int angle)

{

Point centerOfGravity = CenterOfGravity();

double sinA = sin((double)angle \* PI / 180);

double cosA = cos((double)angle \* PI / 180);

double temp;

for (int i = 0; i < \_countSides; i++)

{

temp = \_cords[i].x;

\_cords[i].x = (\_cords[i].x - centerOfGravity.x) \* cosA - (\_cords[i].y - centerOfGravity.y) \* sinA + centerOfGravity.x;

\_cords[i].y = (temp - centerOfGravity.x) \* sinA + (\_cords[i].y - centerOfGravity.y) \* cosA + centerOfGravity.y;

}

}

Rectangle.h

#pragma once

#include "Shape.h"

class Rectangle : public Shape

{

public:

Rectangle(const Point\* cords) : Shape(cords, 4) { TrueShape(); }

double Area() const override;

Point CenterOfGravity() const override;

void TrueShape() const override;

double GetRadius() const override;

};

Rectangle.cpp

#include "Rectangle.h"

#include <stdexcept>

using std::logic\_error;

void Rectangle::TrueShape() const

{

if (\_lengthSide[0] == \_lengthSide[2] && \_lengthSide[1] == \_lengthSide[3] && (\_lengthSide[0] != 0 && \_lengthSide[1] != 0))

{

return;

}

throw logic\_error("WRONG\_RECTANGLE");

}

double Rectangle::Area() const

{

return (ceil(\_lengthSide[0] \* \_lengthSide[1]));

}

Point Rectangle::CenterOfGravity() const

{

Point centerOfGravity;

centerOfGravity.x = (\_cords[0].x + \_cords[1].x + \_cords[2].x + \_cords[3].x) / \_countSides;

centerOfGravity.y = (\_cords[0].y + \_cords[1].y + \_cords[2].y + \_cords[3].y) / \_countSides;

return centerOfGravity;

}

double Rectangle::GetRadius() const

{

double radius;

Point centerOfGravity = CenterOfGravity();

radius = GetLength(centerOfGravity, \_cords[0]);

return radius;

}

Triangle.h

#pragma once

#include "Shape.h"

class Triangle : public Shape

{

public:

Triangle(const Point\* cords) : Shape(cords, 3) { TrueShape(); }

double Area() const override;

Point CenterOfGravity() const override;

void TrueShape() const override;

double GetRadius() const override;

private:

double Perimeter() const;

};

Triangle.cpp

#include "Triangle.h"

#include <stdexcept>

using std::logic\_error;

void Triangle::TrueShape() const

{

if ((\_lengthSide[0] + \_lengthSide[1] > \_lengthSide[2]) && (\_lengthSide[1] + \_lengthSide[2] > \_lengthSide[0])\

&& (\_lengthSide[2] + \_lengthSide[0] > \_lengthSide[1]))

{

return;

}

throw logic\_error("WRONG\_TRIANGLE");

}

double Triangle::Perimeter() const

{

double perimeter = \_lengthSide[0] + \_lengthSide[1] + \_lengthSide[2];

return perimeter;

}

double Triangle::Area() const

{

double area, semiPerimeter;

semiPerimeter = Perimeter() / 2;

area = (round((sqrt(semiPerimeter \* (semiPerimeter - \_lengthSide[0]) \* (semiPerimeter - \_lengthSide[1]) \* (semiPerimeter - \_lengthSide[2])))\*10))/10;

return area;

}

Point Triangle::CenterOfGravity() const

{

Point centerOfGravity;

centerOfGravity.x = ((\_cords[0].x + \_cords[1].x + \_cords[2].x) / 3);

centerOfGravity.y = ((\_cords[0].y + \_cords[1].y + \_cords[2].y) / 3);

return centerOfGravity;

}

double Triangle::GetRadius() const

{

double semiPerimeter = Perimeter() / 2;

double radius;

radius = (\_lengthSide[0] \* \_lengthSide[1] \* \_lengthSide[2]) / (4 \* sqrt(semiPerimeter \* (semiPerimeter - \_lengthSide[0]) \* (semiPerimeter - \_lengthSide[1]) \* (semiPerimeter - \_lengthSide[2])));

return radius;

}

# Набор тестов, охватывающих все режимы работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3