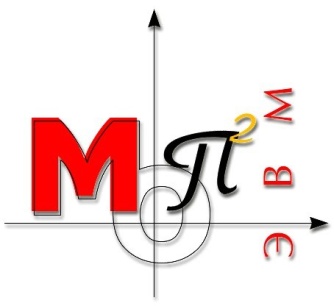
Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждения высшего образования   
«Южный федеральный университет»

Инженерно-технологическая академия

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**

Лабораторная работа №3

по дисциплине

"ООП"

на тему

"Динамический выбор типа объектов"

**Выполнил:**

студент группы КТбо2-6

Пустовой А.В

**Проверил:**

Таганрог, 2020

**Вариант №13**

**Задание:**

В лабораторной работе должна быть создана программа, создающая объекты двух классов (T1, T2), выбранных из таблицы 1 согласно номеру варианта. Эти классы должны быть производными от класса Shape. Перечисленные в таблице классы должны создавать следующие плоские объекты: квадрат, треугольник, прямоугольник, параллелограмм, трапеция, правильный шестиугольник, правильный восьмиугольник (square, triangle, rectangle, parallelogram, trapeze, hexagon, octagon). Для каждого типа фигуры вычисляются площадь, центр тяжести и другие атрибуты [4]; кроме этого, должны быть предусмотрены виртуальные методы Вращения (Rotate) и Перемещения (Move)

Операции над объектами 2˗х классов должны быть такими: - сравнить два объекта по площади - Compare; - определить факт пересечения объектов – IsIntersect; - определить факт включения одного объекта в другой – IsInclude.

**Алгоритмы в решении**

1. Поворот точки вокруг другой точки:

Если вы поворачиваете точку (px, py) вокруг точки (ox, oy) на угол тета, вы получите:

p'x = cos(theta) \* (px-ox) - sin(theta) \* (py-oy) + ox

p'y = sin(theta) \* (px-ox) + cos(theta) \* (py-oy) + oy

это простой способ повернуть точку в 2D.

1. Параллельные прямые:

Прямые параллельны, если их угловые коэффициенты равны

Формула: k = (y2 - y1)/(x2 - x1), где «x» и «y» – координаты двух точек (любых), лежащих на прямой.

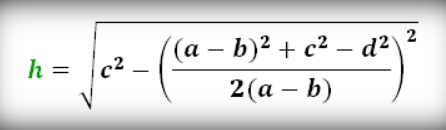
1. Проверка на верность прямоугольника.

У прямоугольника противоположные стороны параллельны и диагонали равны.

1. Центр тяжести прямоугольника находится на пересечении диагоналей и его можно посчитать по формуле y = (y1+y2+y3+y4)/4; x = (x1+x2+x3+x4)/4
2. Площадь прямоугольника находится путем умножения ширины на высоту.
3. Проверка на верность трапеции:

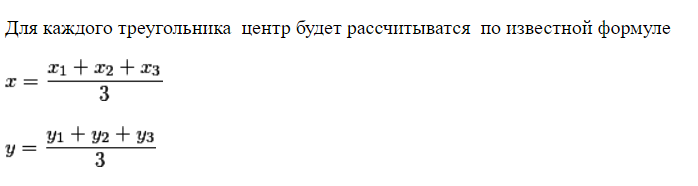
Трапецию можно распознать, если у нее только 2 стороны параллельны, а другие нет

1. Площадь трапеции можно найти по формуле: S=(a+b)/2\*h
2. Высоту трапеции можно найти по формуле:

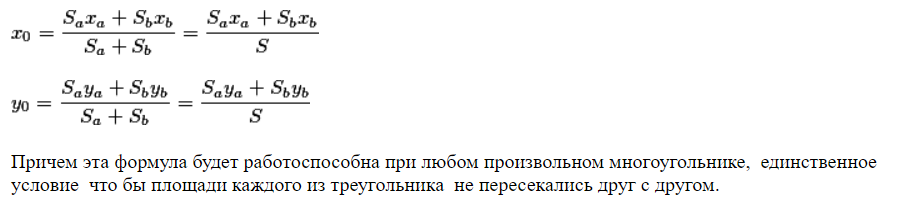


1. Центр тяжести трапеции:

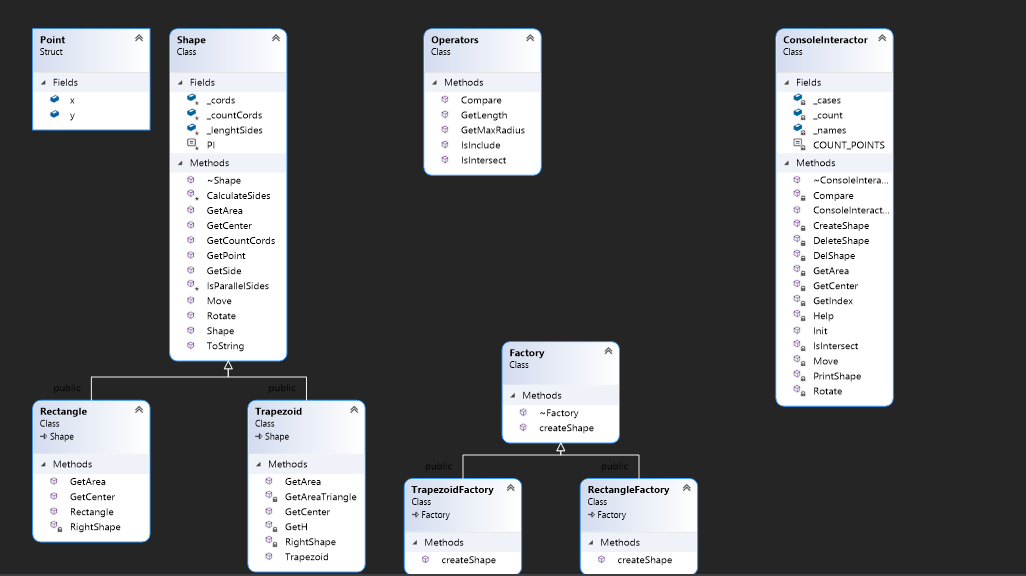
Для начала надо разделить трапеции на 2 треугольника и найти центр тяжести 2ух треугольников



После этого можно будет найти координаты центра тяжести по формуле:



**Диаграмма классов**



**Спецификация классов:**

СonsoleInteractor.h

#include <iostream>

#include "Operators.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 DeleteShape(const string& name) const;

void PrintShape() const;

void DelShape() const;

void Compare() const;

void IsIntersect() const;

void Move() const;

void Rotate() const;

void GetArea() const;

void GetCenter() const;

Shape\*\* \_cases = nullptr;

string\* \_names = nullptr;

int \_count = 0;

const int COUNT\_POINTS = 4;

};

Factory.h

#include "Rectangle.h"

#include "Trapezoid.h"

class Factory

{

public:

virtual Shape\* createShape(Point\* cords) = 0;

virtual ~Factory() {}

};

class RectangleFactory : public Factory

{

public:

Shape\* createShape(Point\* cords) override

{

return new Rectangle(cords);

}

};

class TrapezoidFactory : public Factory

{

public:

Shape\* createShape(Point\* cords) override

{

return new Trapezoid(cords);

}

};

Operators.h

#include "Shape.h"

class Operators

{

public:

int Compare(Shape\* shape1, Shape\* shape2) const;

bool IsIntersect(Shape\* shape1, Shape\* shape2) const;

bool IsInclude(Shape\* shape1, Shape\* shape2) const;

double GetLength(Point& point1, Point& point2) const;

double GetMaxRadius(Shape\* shape) const;

};

Rectangle.h

#include "Shape.h"

#include <stdexcept>

class Rectangle : public Shape

{

public:

Rectangle(Point\* cords);

double GetArea() const override;

Point GetCenter() const override;

private:

bool RightShape(Point\* cords) const;

};

Shape.h

#pragma once

#include <stdexcept>

#include <cmath>

#include <string>

struct Point

{

double x;

double y;

};

class Shape

{

public:

Shape() : \_cords(nullptr), \_countCords(0), \_lenghtSides(nullptr) {}

virtual ~Shape();

int GetCountCords() const;

double GetSide(const int index) const;

void Move(const char direction, const double count);

std::string ToString() const;

Point& GetPoint(const int index) const;

void Rotate(const int angle);

virtual double GetArea() const = 0;

virtual Point GetCenter() const = 0;

protected:

Point\* \_cords;

int \_countCords;

double\* \_lenghtSides;

const double PI = 3.14159265358979323846;

void CalculateSides();

bool IsParallelSides(const Point& a1, const Point& a2, const Point& b1, const Point& b2) const;

};

Trapezoid.h

#include "Shape.h"

class Trapezoid : public Shape

{

public:

Trapezoid(Point\* cords);

double GetArea() const override;

Point GetCenter() const override;

private:

bool RightShape(Point\* cords) const;

double GetH() const;

double GetAreaTriangle(const Point& dot1, const Point& dot2, const Point& dot3) const;

};

**Листинг программы**

СonsoleInteractor.h

#include <iostream>

#include "Operators.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 DeleteShape(const string& name) const;

void PrintShape() const;

void DelShape() const;

void Compare() const;

void IsIntersect() const;

void Move() const;

void Rotate() const;

void GetArea() const;

void GetCenter() const;

Shape\*\* \_cases = nullptr;

string\* \_names = nullptr;

int \_count = 0;

const int COUNT\_POINTS = 4;

};

Factory.h

#include "Rectangle.h"

#include "Trapezoid.h"

class Factory

{

public:

virtual Shape\* createShape(Point\* cords) = 0;

virtual ~Factory() {}

};

class RectangleFactory : public Factory

{

public:

Shape\* createShape(Point\* cords) override

{

return new Rectangle(cords);

}

};

class TrapezoidFactory : public Factory

{

public:

Shape\* createShape(Point\* cords) override

{

return new Trapezoid(cords);

}

};

ConsoleInteractor.cpp

#include <iostream>

#include "ConsoleInteractor.h"

using std::cin;

using std::cout;

ConsoleInteractor::~ConsoleInteractor()

{

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

{

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

{

delete \_cases[i];

}

}

delete[] \_names;

delete[] \_cases;

}

ConsoleInteractor::ConsoleInteractor()

{

cout << "Enter space for Shapes\n>> ";

cin >> \_count;

\_names = new string[\_count];

\_cases = new Shape\* [\_count];

}

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

{

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

{

if (name == \_names[i])

{

return i;

}

}

return -1;

}

void ConsoleInteractor::Help() const

{

cout << "\"1\" - Create new Shape\n\"2\" - Delete Shape\n\"3\" - Help\n\"4\" - Compare \n\"5\" - IsIntersect\n\

\"6\" - Print Shape\n\"7\" - Move\n\"8\" - Rotate\n\"9\" - Get Area\n\"10\" - Get Center\n\"11\" - Exit\n";

}

void ConsoleInteractor::CreateShape() const

{

RectangleFactory rectangleFactory;

TrapezoidFactory trapezoidFactory;

cout << "Enter name of Shape:\n>> ";

string name;

cin >> name;

int type;

cout << "Is it Rectangle or Trapezoid (1 or 2):\n>> ";

cin >> type;

Point\* points = new Point[COUNT\_POINTS];

cout << "Enter points:\n";

for (int i = 0; i < COUNT\_POINTS; i++)

{

cout << "\n" << i + 1 << ") x: \n>> ";

cin >> points[i].x;

cout << "\n" << i + 1 << ") y: \n>> ";

cin >> points[i].y;

}

bool isEnoughtSpace = false;

int index;

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

{

if (\_names[i].empty())

{

isEnoughtSpace = true;

index = i;

}

}

if (isEnoughtSpace)

{

Shape\* newElement = nullptr;

try

{

if (type == 1)

{

newElement = rectangleFactory.createShape(points);

}

else if (type == 2)

{

newElement = trapezoidFactory.createShape(points);

}

\_names[index] = name;

\_cases[index] = newElement;

}

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

{

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

}

}

else

{

cout << "You don't have enought space, delete something\n";

}

delete[] points;

}

void ConsoleInteractor::DeleteShape(const string& name) const

{

int index = GetIndex(name);

if (index == -1)

{

cout << "Can't find pair with this name\n";

}

else

{

\_names[index] = "";

delete \_cases[index];

\_cases[index] = nullptr;

}

}

void ConsoleInteractor::PrintShape() const

{

string name;

cout << "Enter name of pair to print\n>> ";

cin >> name;

int index = GetIndex(name);

if (index != -1)

{

cout << \_cases[index]->ToString() << "\n";

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::DelShape() const

{

string name;

cout << "Enter name of pair to delete\n>> ";

cin >> name;

DeleteShape(name);

cout << "Complete\n";

}

void ConsoleInteractor::Compare() const

{

Operators operators;

string name, name2;

cout << "Enter name of first shape\n>> ";

cin >> name;

cout << "Enter name of second shape\n>> ";

cin >> name2;

int indexName1 = GetIndex(name);

int indexName2 = GetIndex(name2);

if (indexName1 != -1 && indexName2 != -1)

{

switch (operators.Compare(\_cases[indexName1], \_cases[indexName2]))

{

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 << "Wrong name\n";

}

}

void ConsoleInteractor::Move() const

{

string name;

char side;

double count;

cout << "Enter name of shape\n>> ";

cin >> name;

int indexName1 = GetIndex(name);

if (indexName1 != -1)

{

cout << "Which side?(u-up, l-left, r-right, d-down)\n>> ";

cin >> side;

cout << "How much\n>> ";

cin >> count;

\_cases[indexName1]->Move(side, count);

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::Rotate() const

{

string name;

int angle;

cout << "Enter name of shape\n>> ";

cin >> name;

int indexName1 = GetIndex(name);

if (indexName1 != -1)

{

cout << "Enter angle\n>> ";

cin >> angle;

\_cases[indexName1]->Rotate(angle);

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::GetArea() const

{

string name;

cout << "Enter name of shape\n>> ";

cin >> name;

int indexName1 = GetIndex(name);

if (indexName1 != -1)

{

cout << "Area of this shape is " << \_cases[indexName1]->GetArea() << "\n";

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::GetCenter() const

{

string name;

cout << "Enter name of shape\n>> ";

cin >> name;

int indexName1 = GetIndex(name);

if (indexName1 != -1)

{

Point center = \_cases[indexName1]->GetCenter();

cout << "Center of this shape: x = " << center.x << " y = " << center.y << "\n";

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::IsIntersect() const

{

Operators operators;

string name, name2;

cout << "Enter name of first shape\n>> ";

cin >> name;

cout << "Enter name of second shape\n>> ";

cin >> name2;

int indexName1 = GetIndex(name);

int indexName2 = GetIndex(name2);

if (indexName1 != -1 && indexName2 != -1)

{

if (operators.IsIntersect(\_cases[indexName1], \_cases[indexName2]))

{

cout << "Shapes intersect\n";

}

else

{

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

}

}

else

{

cout << "Wrong name\n";

}

}

void ConsoleInteractor::Init() const

{

bool flag = true;

while (flag)

{

int code;

cout << "Enter \"3\" - Help\n";

cout << ">> ";

cin >> code;

switch (code)

{

case 1:

CreateShape();

break;

case 2:

DelShape();

break;

case 3:

Help();

break;

case 4:

Compare();

break;

case 5:

IsIntersect();

break;

case 6:

PrintShape();

break;

case 7:

Move();

break;

case 8:

Rotate();

break;

case 9:

GetArea();

break;

case 10:

break;

case 11:

flag = false;

break;

}

}

}

Operators.h

#include "Shape.h"

class Operators

{

public:

int Compare(Shape\* shape1, Shape\* shape2) const;

bool IsIntersect(Shape\* shape1, Shape\* shape2) const;

bool IsInclude(Shape\* shape1, Shape\* shape2) const;

double GetLength(Point& point1, Point& point2) const;

double GetMaxRadius(Shape\* shape) const;

};

Operators.cpp

#include "Operators.h"

int Operators::Compare(Shape\* shape1, Shape\* shape2) const

{

int res;

double area1 = shape1->GetArea();

double area2 = shape2->GetArea();

if (area1 > area2)

{

return 1;

}

else if (area1 < area2)

{

return -1;

}

return 0;

}

bool Operators::IsIntersect(Shape\* shape1, Shape\* shape2) const

{

double radius1 = GetMaxRadius(shape1);

double radius2 = GetMaxRadius(shape2);

Point center1 = shape1->GetCenter();

Point center2 = shape2->GetCenter();

double length = GetLength(center1, center2);

if (length > radius1 + radius2 || length <= abs(radius1 - radius2))

{

return false;

}

return true;

}

bool Operators::IsInclude(Shape\* shape1, Shape\* shape2) const

{

double radius1 = GetMaxRadius(shape1);

double radius2 = GetMaxRadius(shape2);

Point center1 = shape1->GetCenter();

Point center2 = shape2->GetCenter();

double length = GetLength(center1, center2);

if (length <= abs(radius1 - radius2))

{

return true;

}

return false;

}

double Operators::GetLength(Point& point1, Point& point2) const

{

return sqrt(pow(point2.x - point1.x, 2) + pow(point2.y - point1.y, 2));

}

double Operators::GetMaxRadius(Shape\* shape) const

{

double max = 0;

Point center = shape->GetCenter();

for (int i = 0; i < shape->GetCountCords(); i++)

{

double lenght = GetLength(center, shape->GetPoint(i));

if (max < lenght)

{

max = lenght;

}

}

return max;

}

Rectangle.h

#include "Shape.h"

#include <stdexcept>

class Rectangle : public Shape

{

public:

Rectangle(Point\* cords);

double GetArea() const override;

Point GetCenter() const override;

private:

bool RightShape(Point\* cords) const;

};

Rectangle.cpp

#include "Rectangle.h"

using std::logic\_error;

Rectangle::Rectangle(Point\* cords)

{

\_countCords = 4;

if (!RightShape(cords))

{

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

}

\_cords = new Point[\_countCords];

\_lenghtSides = new double[\_countCords];

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

CalculateSides();

}

double Rectangle::GetArea() const

{

return \_lenghtSides[0] \* \_lenghtSides[1];

}

Point Rectangle::GetCenter() const

{

Point center;

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

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

return center;

}

bool Rectangle::RightShape(Point\* cords) const

{

bool IsParallel1 = IsParallelSides(cords[0], cords[1], cords[2], cords[3]);

bool IsParallel2 = IsParallelSides(cords[1], cords[2], cords[3], cords[0]);

double diagonal1 = sqrt(pow(cords[0].x - cords[2].x, 2) + pow(cords[0].y - cords[2].y, 2));

double diagonal2 = sqrt(pow(cords[1].x - cords[3].x, 2) + pow(cords[1].y - cords[3].y, 2));

if (diagonal1 == diagonal2 && IsParallel1 && IsParallel2)

{

return true;

}

return false;

}

Shape.h

#pragma once

#include <stdexcept>

#include <cmath>

#include <string>

struct Point

{

double x;

double y;

};

class Shape

{

public:

Shape() : \_cords(nullptr), \_countCords(0), \_lenghtSides(nullptr) {}

virtual ~Shape();

int GetCountCords() const;

double GetSide(const int index) const;

void Move(const char direction, const double count);

std::string ToString() const;

Point& GetPoint(const int index) const;

void Rotate(const int angle);

virtual double GetArea() const = 0;

virtual Point GetCenter() const = 0;

protected:

Point\* \_cords;

int \_countCords;

double\* \_lenghtSides;

const double PI = 3.14159265358979323846;

void CalculateSides();

bool IsParallelSides(const Point& a1, const Point& a2, const Point& b1, const Point& b2) const;

};

Shape.cpp

#include "Shape.h"

Shape::~Shape()

{

delete[] \_lenghtSides;

delete[] \_cords;

}

int Shape::GetCountCords() const

{

return \_countCords;

}

void Shape::Move(const char direction, const double count)

{

switch (direction)

{

case 'u':

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

{

\_cords[i].y += count;

}

break;

case 'd':

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

{

\_cords[i].y -= count;

}

break;

case 'l':

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

{

\_cords[i].x -= count;

}

break;

case 'r':

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

{

\_cords[i].x += count;

}

break;

default:

break;

}

}

Point& Shape::GetPoint(const int index) const

{

if (index < \_countCords && index >= 0)

{

return \_cords[index];

}

else

{

throw std::out\_of\_range("OUT\_OF\_RANGE");

}

}

void Shape::CalculateSides()

{

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

{

double x1 = \_cords[i % \_countCords].x;

double x2 = \_cords[(i + 1) % \_countCords].x;

double y1 = \_cords[i % \_countCords].y;

double y2 = \_cords[(i + 1) % \_countCords].y;

\_lenghtSides[i] = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));

}

}

void Shape::Rotate(const int angle)

{

Point center = GetCenter();

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

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

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

{

\_cords[i].x -= center.x;

\_cords[i].y -= center.y;

double newx = \_cords[i].x \* cosA - \_cords[i].y \* sinA;

double newy = \_cords[i].x \* sinA + \_cords[i].y \* cosA;

\_cords[i].x = newx + center.x;

\_cords[i].y = newy + center.y;

}

}

double Shape::GetSide(const int index) const

{

if (index < \_countCords && index >= 0)

{

return \_lenghtSides[index];

}

else

{

throw std::out\_of\_range("OUT\_OF\_RANGE");

}

}

std::string Shape::ToString() const

{

std::string result = "";

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

{

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

}

return result;

}

bool Shape::IsParallelSides(const Point& a1, const Point& a2, const Point& b1, const Point& b2) const

{

double k1 = (a2.y - a1.y) / (a2.x - a1.x);

double k2 = (b2.y - b1.y) / (b2.x - b1.x);

if (k1 == k2 || !(a2.x - a1.x) && !(b2.x - b1.x))

{

return true;

}

return false;

}

Trapezoid.h

#include "Shape.h"

class Trapezoid : public Shape

{

public:

Trapezoid(Point\* cords);

double GetArea() const override;

Point GetCenter() const override;

private:

bool RightShape(Point\* cords) const;

double GetH() const;

double GetAreaTriangle(const Point& dot1, const Point& dot2, const Point& dot3) const;

};

Trapezoid.cpp

#include "Trapezoid.h"

#include <stdexcept>

using std::logic\_error;

Trapezoid::Trapezoid(Point\* cords)

{

\_countCords = 4;

if (!RightShape(cords))

{

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

}

\_cords = new Point[\_countCords];

\_lenghtSides = new double[\_countCords];

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

CalculateSides();

}

double Trapezoid::GetArea() const

{

double h = GetH();

return (\_lenghtSides[1] + \_lenghtSides[3]) / 2 \* h;

}

Point Trapezoid::GetCenter() const

{

double centerTriangleX1 = (\_cords[0].x + \_cords[1].x + \_cords[2].x) / 3;

double centerTriangleY1 = (\_cords[0].y + \_cords[1].y + \_cords[2].y) / 3;

double centerTriangleX2 = (\_cords[2].x + \_cords[3].x + \_cords[0].x) / 3;

double centerTriangleY2 = (\_cords[2].y + \_cords[3].y + \_cords[0].y) / 3;

double areaTriangle1 = GetAreaTriangle(\_cords[0], \_cords[1], \_cords[2]);

double areaTriangle2 = GetAreaTriangle(\_cords[2], \_cords[3], \_cords[0]);

Point center;

center.x = (areaTriangle1 \* centerTriangleX1 + areaTriangle2 \* centerTriangleX2) / (areaTriangle1 + areaTriangle2);

center.y = (areaTriangle1 \* centerTriangleY1 + areaTriangle2 \* centerTriangleY2) / (areaTriangle1 + areaTriangle2);

return center;

}

double Trapezoid::GetH() const

{

double c = \_lenghtSides[0];

double a = \_lenghtSides[1];

double d = \_lenghtSides[2];

double b = \_lenghtSides[3];

return sqrt(c \* c - ((pow(b - a, 2) + c \* c - d \* d) / (2 \* (b - a))));

}

bool Trapezoid::RightShape(Point\* cords) const

{

bool IsParallel1 = IsParallelSides(cords[0], cords[1], cords[2], cords[3]);

bool IsParallel2 = IsParallelSides(cords[1], cords[2], cords[3], cords[0]);

if (IsParallel1 && !IsParallel2 || !IsParallel1 && IsParallel2)

{

return true;

}

return false;

}

double Trapezoid::GetAreaTriangle(const Point& dot1, const Point& dot2, const Point& dot3) const

{

return abs(0.5 \* ((dot1.x - dot3.x) \* (dot2.y - dot3.y) - (dot1.y - dot3.y) \* (dot2.x - dot3.x)));

}

Main.cpp

#include <iostream>

#include "ConsoleInteractor.h"

int main()

{

ConsoleInteractor console;

console.Init();

return 0;

}

**Вывод**

Я ознакомился с механизмом выбора типа создаваемых объектов во время выполнения программы.