**Лабораторная работа №3**

**Динамический выбор типа объектов**

**Цель работы**

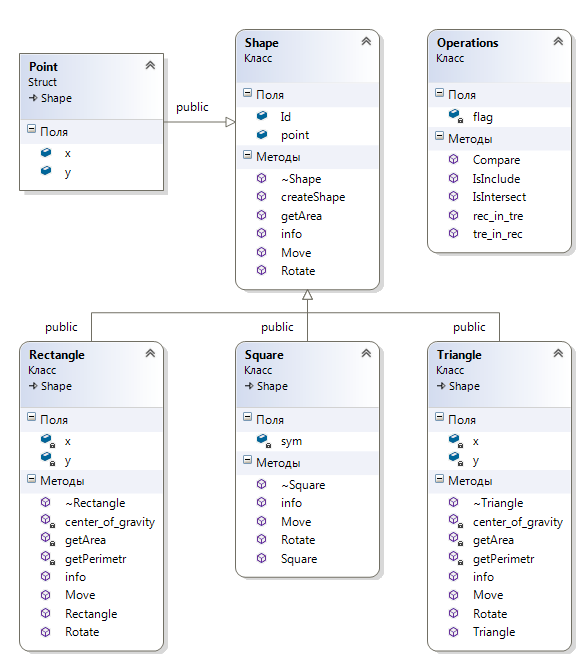
Ознакомление с механизмом выбора типа создаваемых объектов во время выполнения программы.

**Формулировка задачи**

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: квадрат, треугольник, прямоугольник, параллелограмм, трапеция, правильный шестиугольник, правильный восьмиугольник. Для каждого типа фигуры вычисляются площадь, центр тяжести и другие атрибуты, а также должны быть предусмотрены виртуальные методы Вращения (Rotate) и Перемещения (Move). Треугольник и Прямоугольник.

**Диаграммы классов**



**Листинг программы lab3\_oop.cpp**

#include "Console.h"

int main()

{

setlocale(LC\_ALL, "RUS");

Console first;

first.init();

}

**Листинг программы Console.h**

#pragma once

#include <iostream>

#include <cmath>

//#include <vld.h>

using namespace std;

#include "Figure.h"

// в библиотеке нет константы

#define M\_PI 3.14159265358979323846

class Console{

public:

Point\* enter\_Point(char c);

int init();

};

**Листинг программы Console.cpp**

#include "Console.h"

Point\* Console::enter\_Point(char c) {

if (c == 'T')

{

Point\* ptr = new Point[3];

cout << "enter 1 vertex\n";

cin >> ptr[0].x >> ptr[0].y;

cout << "enter 2 vertex\n";

cin >> ptr[1].x >> ptr[1].y;

cout << "enter 3 vertex\n";

cin >> ptr[2].x >> ptr[2].y;

return ptr;

}

else if (c == 'R')

{

Point\* ptr = new Point[4];

cout << "enter 1 vertex\n";

cin >> ptr[0].x >> ptr[0].y;

cout << "enter 2 vertex\n";

cin >> ptr[1].x >> ptr[1].y;

cout << "enter 3 vertex\n";

cin >> ptr[2].x >> ptr[2].y;

cout << "enter 4 vertex\n";

cin >> ptr[3].x >> ptr[3].y;

return ptr;

}

return nullptr;

}

int Console::init()

{

// Объекты базового класса и класса Operations, symbol - символ фигуры

Shape\* ptr1 = 0, \* ptr2 = 0;

Point\* pt;

Operations op;

char symbol;

// Ввод типа фигуры

cout << "Выберете тип фигуры: ";

cin >> symbol;

// Вывод информации о фигуре

ptr1 = Shape::createShape(symbol, enter\_Point(symbol));

cout << ptr1->info();

// Ввод типа фигуры

cout << "Выберете тип фигуры: ";

cin >> symbol;

// Вывод информации о фигуре

ptr2 = Shape::createShape(symbol, enter\_Point(symbol));

cout << ptr2->info();

// Вызов операций сравнения, включения и пересечения двух фигур

op.Compare(ptr1, ptr2);

op.IsIntersect(ptr1, ptr2);

op.IsInclude(ptr1, ptr2);

double angle;

cout << "введите 1 угол поворота: ";

cin >> angle;

ptr1->Rotate(angle);

cout << ptr1->info();

cout << "введите 2 угол поворота: ";

cin >> angle;

ptr2->Rotate(angle);

cout << ptr2->info();

cout << "введите 1 смещение: ";

double x, y;

cin >> x >> y;

ptr1->Move(x, y);

cout << ptr1->info();

cout << "введите 2 смещение: ";

cin >> x >> y;

ptr1->Move(x, y);

cout << ptr2->info();

// Очищение памяти

delete ptr1;

delete ptr2;

return 0;

}

**Листинг программы Figure.h**

#include "Rectangle.h"

#include "Square.h"

#include "Triangle.h"

using namespace std;

class Operations

{

private:

int flag = 0;

public:

// Метод сравнения площадей

void Compare(Shape\* p1, Shape\* p2);

// Метод проверки пересечения площадей

void IsIntersect(Shape\* p1, Shape\* p2);

// Метод проверки включения фигур друг в друга

void IsInclude(Shape\* p1, Shape\* p2);

// Метод проверки включения прямоугольника в треугольник

bool rec\_in\_tre(Shape\* p1, Shape\* p2);

// Метод проверки включения треугольника в прямоугольник

bool tre\_in\_rec(Point A, Point B, Point C, Point P);

};

**Листинг программы Figure.cpp**

#include "Figure.h"

// Метод сравнения площадей

void Operations::Compare(Shape\* p1, Shape\* p2)

{

// Сравнение площади первой и второй фигуры

if (p1->getArea() > p2->getArea())

cout << "Площадь первой фигуры больше площади второй фигуры" << endl;

else if (p1->getArea() < p2->getArea())

cout << "Площадь второй фигуры больше площали первой фигуры" << endl;

else

cout << "Площади фигур равны" << endl;

}

// Метод проверки пересечения площадей

void Operations::IsIntersect(Shape\* p1, Shape\* p2)

{

if (!rec\_in\_tre(p1, p2) && flag == 1)

cout << "Фигуры пересекаются" << endl;

else if ((!tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[0]) || !tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[0]) || !tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[0])) && flag == 1)

if ((!tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[1]) || !tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[1]) || !tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[1])) && flag == 1)

if ((!tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[2]) || !tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[2]) || !tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[2])) && flag == 1)

cout << "Фигуры пересекаются" << endl;

else cout << "Фигуры не пересекаются" << endl;

else cout << "Фигуры не пересекаются" << endl;

else cout << "Фигуры не пересекаются" << endl;

flag = 0;

}

// Метод проверки включения фигур друг в друга

void Operations::IsInclude(Shape\* p1, Shape\* p2)

{

// Если площадь треугольника больше площади прямоугольника

if (p1->getArea() >= p2->getArea())

{

// Если функция rec\_in\_tre вернула 1, то прямоугольник заключён в треугольник, иначе фигуры не заключены

if (rec\_in\_tre(p1, p2))

cout << "Прямоугольник заключён в треугольник" << endl;

else cout << "Фигуры не заключены" << endl;

}

// Если площадь треугольника меньше площади прямоугольника

else if (p1->getArea() < p2->getArea())

{

// Если функция tre\_in\_rec вернула во всех проверках 1, то треугольник заключён в прямоугольник, иначе фигуры не заключены

if (tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[0]) || tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[0]) || tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[0]))

if (tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[1]) || tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[1]) || tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[1]))

if (tre\_in\_rec(p2->point[0], p2->point[1], p2->point[2], p1->point[2]) || tre\_in\_rec(p2->point[1], p2->point[2], p2->point[3], p1->point[2]) || tre\_in\_rec(p2->point[2], p2->point[3], p2->point[0], p1->point[2]))

cout << "Треугольник заключён в прямоугольник" << endl;

else cout << "Фигуры не заключены" << endl;

else cout << "Фигуры не заключены" << endl;

else cout << "Фигуры не заключены" << endl;

}

else

cout << "Фигуры не заключены" << endl;

flag = 0;

}

// Метод проверки включения прямоугольника в треугольник

bool Operations::rec\_in\_tre(Shape\* p1, Shape\* p2)

{

double a, b, c;

int fl = 1;

// Цикл проходит по каждой точке прямоугольника и проверяет её на заключённость в треугольник

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

{

a = (p1->point[0].x - p2->point[i].x) \* (p1->point[1].y - p1->point[0].y) - (p1->point[1].x - p1->point[0].x) \* (p1->point[0].y - p2->point[i].y);

b = (p1->point[1].x - p2->point[i].x) \* (p1->point[2].y - p1->point[1].y) - (p1->point[2].x - p1->point[1].x) \* (p1->point[1].y - p2->point[i].y);

c = (p1->point[2].x - p2->point[i].x) \* (p1->point[0].y - p1->point[2].y) - (p1->point[0].x - p1->point[2].x) \* (p1->point[2].y - p2->point[i].y);

if ((a >= 0 && b >= 0 && c >= 0) || (a <= 0 && b <= 0 && c <= 0))

flag = 1;

else

fl = 0;

}

if (flag && fl)

return 1;

else

return 0;

}

// Метод проверки включения треугольника в прямоугольник

bool Operations::tre\_in\_rec(Point A, Point B, Point C, Point P)

{

double a, b, c;

int fl = 1;

// Цикл проходит по каждой точке треугольника и проверяет её на заключённость в прямоугольник

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

{

a = (A.x - P.x) \* (B.y - A.y) - (B.x - A.x) \* (A.y - P.y);

b = (B.x - P.x) \* (C.y - B.y) - (C.x - B.x) \* (B.y - P.y);

c = (C.x - P.x) \* (A.y - C.y) - (A.x - C.x) \* (C.y - P.y);

if ((a >= 0 && b >= 0 && c >= 0) || (a <= 0 && b <= 0 && c <= 0))

flag = 1;

else

fl = 0;

}

if (flag && fl)

return 1;

else

return 0;

}

**Листинг программы Rectangle.h**

#pragma once

#include <math.h>

#include "Shape.h"

using std::to\_string;

class Rectangle : public Shape

{

private:

//Центр Масс

double x, y;

double getArea();

double getPerimetr();

void center\_of\_gravity();

public:

Rectangle(Point\* pt);

void Rotate(double rotation\_angle);

void Move(double move\_x, double move\_y);

;

std::string info();

;

~Rectangle();

};

**Листинг программы Rectangle.cpp**

#include "Rectangle.h"

double Rectangle::getArea()

{

double line1 = abs(point[0].x - point[3].x);

double line2 = abs(point[0].y - point[1].y);

return (line1 \* line2);

}

double Rectangle::getPerimetr()

{

double line1 = abs(point[0].x - point[3].x);

double line2 = abs(point[0].y - point[1].y);

return 2 \* (line1 + line2);

}

void Rectangle::center\_of\_gravity()

{

double temp\_x = (abs(point[0].x - point[3].x) / 2);

double temp\_y = (abs(point[0].y - point[1].y) / 2);

double gravity\_x = point[0].x + temp\_x;

double gravity\_y = point[0].y + temp\_y;

x = gravity\_x;

y = gravity\_y;

}

Rectangle::Rectangle(Point\* pt)

{

Id = 'R';

point = new Point[4];

point[0] = pt[0];

point[1] = pt[1];

point[2] = pt[2];

point[3] = pt[3];

}

void Rectangle::Rotate(double rotation\_angle)

{

double rotate\_x, rotate\_y;

/\*cout << "Введите угол поворота : ";

cin >> rotation\_angle;\*/

rotate\_x = (cos(rotation\_angle) \* point[0].x) - (sin(rotation\_angle) \* point[0].y);

rotate\_y = (sin(rotation\_angle) \* point[0].x) + (cos(rotation\_angle) \* point[0].y);

// Новые координаты точки (х1, y1)

point[0].x = rotate\_x;

point[0].y = rotate\_y;

rotate\_x = (cos(rotation\_angle) \* point[1].x) - (sin(rotation\_angle) \* point[1].y);

rotate\_y = (sin(rotation\_angle) \* point[1].x) + (cos(rotation\_angle) \* point[1].y);

// Новые координаты точки (х2, y2)

point[1].x = rotate\_x;

point[1].y = rotate\_y;

rotate\_x = (cos(rotation\_angle) \* point[2].x) - (sin(rotation\_angle) \* point[2].y);

rotate\_y = (sin(rotation\_angle) \* point[2].x) + (cos(rotation\_angle) \* point[2].y);

// Новые координаты точки (х3, y3)

point[2].x = rotate\_x;

point[2].y = rotate\_y;

rotate\_x = (cos(rotation\_angle) \* point[3].x) - (sin(rotation\_angle) \* point[3].y);

rotate\_y = (sin(rotation\_angle) \* point[3].x) + (cos(rotation\_angle) \* point[3].y) + 100;

// Новые координаты точки (х4, y4)

point[3].x = rotate\_x;

point[3].y = rotate\_y;

}

void Rectangle::Move(double move\_x, double move\_y)

{

if (move\_x)

{

point[0].x = point[0].x + move\_x;

point[1].x = point[1].x + move\_x;

point[2].x = point[2].x + move\_x;

point[3].x = point[3].x + move\_x;

}

if (move\_y)

{

point[0].y = point[0].y + move\_y;

point[1].y = point[1].y + move\_y;

point[2].y = point[2].y + move\_y;

point[3].y = point[3].y + move\_y;

}

}

;

std::string Rectangle::info() {

std::string s = "";

s += "Прямоугольник\n";

s += "Координаты точек прямоугольника: \n";

s += "x1 = "; s += to\_string(point[0].x); s += ";"; s += " y1 = "; s += to\_string(point[0].y); s += "\n";

s += "x2 = "; s += to\_string(point[1].x); s += ";"; s += " y2 = "; s += to\_string(point[1].y); s += "\n";

s += "x3 = "; s += to\_string(point[2].x); s += ";"; s += " y3 = "; s += to\_string(point[2].y); s += "\n";

s += "x4 = "; s += to\_string(point[3].x); s += ";"; s += " y4 = "; s += to\_string(point[3].y); s += "\n";

s += "Площадь прямоугольника : "; s += to\_string(getArea()); s += "\n";

s += "Периметр прямоугольника : "; s += to\_string(getPerimetr()); s += "\n";

center\_of\_gravity();

s += "Центр тяжести : x = "; s += to\_string(x); s += ", y = "; s += to\_string(y); s += "\n";

return s;

}

;

Rectangle::~Rectangle()

{

delete[] point;

}

**Листинг программы Triangle.h**

#pragma once

#include <math.h>

#include "Shape.h"

using namespace std;

class Triangle : public Shape

{

private:

double x, y;

double getArea() ;

double getPerimetr() ;

void center\_of\_gravity() ;

public:

Triangle(Point \*pt);

void Rotate(double rotation\_angle);

void Move(double move\_x, double move\_y);

std::string info() ;

~Triangle();

};

**Листинг программы Triangle.cpp**

#include "Triangle.h"

using namespace std;

double Triangle::getArea()

{

// S = 1/2 \* a \* b

return 0.5 \* abs(((point[0].x - point[2].x) \* (point[1].y - point[2].y)) - ((point[1].x - point[2].x) \* (point[0].y - point[2].y)));

}

double Triangle::getPerimetr()

{

double line1 = sqrt(pow((point[1].x - point[0].x), 2) + pow((point[1].y - point[0].y), 2)); // Между 1 и 2 точкой

double line2 = sqrt(pow((point[2].x - point[0].x), 2) + pow((point[2].y - point[0].y), 2)); // Между 1 и 3 точкой

double line3 = sqrt(pow((point[2].x - point[1].x), 2) + pow((point[2].y - point[1].y), 2)); // Между 2 и 3 точкой

return line1 + line2 + line3;

}

void Triangle::center\_of\_gravity()

{

double gravity\_x = abs((point[0].x + point[1].x + point[2].x) / 3);

double gravity\_y = abs((point[0].y + point[1].y + point[2].y) / 3);

x = gravity\_x;

y = gravity\_y;

}

/\*

Triangle::Triangle(Point \*pt)

{

// Id треугольника - T

Id = 'T';

point = new Point[3];

point[0] = pt[0];

point[1] = pt[1];

point[2] = pt[2];

}

\*/

Triangle::Triangle(Point\* pt)

{

Id = 'T';

point = new Point[3];

point[0] = pt[0];

point[1] = pt[1];

point[2] = pt[2];

}

void Triangle::Rotate(double rotation\_angle)

{

//double rotation\_angle;

double rotate\_x, rotate\_y;

cin >> rotation\_angle;

rotate\_x = (cos(rotation\_angle) \* point[0].x) - (sin(rotation\_angle) \* point[0].y);

rotate\_y = (sin(rotation\_angle) \* point[0].x) + (cos(rotation\_angle) \* point[0].y);

point[0].x = rotate\_x;

point[0].y = rotate\_y;

rotate\_x = (cos(rotation\_angle) \* point[1].x) - (sin(rotation\_angle) \* point[1].y);

rotate\_y = (sin(rotation\_angle) \* point[1].x) + (cos(rotation\_angle) \* point[1].y);

point[1].x = rotate\_x;

point[1].y = rotate\_y;

rotate\_x = (cos(rotation\_angle) \* point[2].x) - (sin(rotation\_angle) \* point[2].y);

rotate\_y = (sin(rotation\_angle) \* point[2].x) + (cos(rotation\_angle) \* point[2].y);

point[2].x = rotate\_x;

point[2].y = rotate\_y;

}

void Triangle::Move(double move\_x, double move\_y) {

if (move\_x)

{

point[0].x = point[0].x + move\_x;

point[1].x = point[1].x + move\_x;

point[2].x = point[2].x + move\_x;

}

if (move\_y)

{

point[0].y = point[0].y + move\_y;

point[1].y = point[1].y + move\_y;

point[2].y = point[2].y + move\_y;

}

};

string Triangle::info()

{

string s = "";

s += "Треугольник\n";

s += "Координаты точек треугольника: \n";

s += "x1 = "; s += to\_string(point[0].x); s += "; y1 = "; s += to\_string(point[0].y); s += "\n";

s += "x2 = "; s += to\_string(point[1].x); s += "; y2 = "; s += to\_string(point[1].y); s += "\n";

s += "x3 = "; s += to\_string(point[2].x); s += "; y3 = "; s += to\_string(point[2].y); s += "\n";

s += "Площадь треугольника : "; s += to\_string(getArea()); s += "\n";

s += "Периметр треугольника : "; s += to\_string(getPerimetr()); s += "\n";

center\_of\_gravity();

s += "Центр тяжести : x = "; s += to\_string(x); s += ", y = "; s += to\_string(y); s += "\n";

return s;

}

Triangle::~Triangle()

{

delete[] point;

}

**Листинг программы Sqare.h**

#pragma once

#include <math.h>

#include "Shape.h"

class Square : public Shape

{

public:

Square( )

{

}

~Square() { delete[] point; };

std::string info() { return "обработка не предусмотрена"; };

void Rotate() {};

void Move() {};

};

**Листинг программы Point.h**

struct Point {

double x;

double y;

};

**Заключение**

В процессе выполнения лабораторной работы №3 мы ознакомились с отношением наследования и иерархией классов. Изучили, что такое виртуальные методы, как происходит наследование и какая разница между модификаторами доступа и наследования.

Авторы Преподаватель: Тарасов С.А.

Шутенко Остап Русланович Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись \_\_\_\_\_\_\_ Дата: \_\_.\_\_.2020