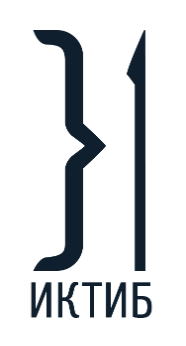
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ



**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Динамический выбор типа объектов»**

*Вариант № 7*

Выполнил(и):

Студенты группы

КТбо2-1

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ус Д. Д. |
|  | *подпись* |  |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Михайлов А. Г. |
|  | *подпись* |  |

Проверил:

ассистент кафедры

МОП ЭВМ

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Жиглатый А.А. |
|  | *подпись* |  |

Оценка

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

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

ד

ינТаганрог 2021

1. **Постановка задачи**

Создание объекта класса во время выполнения программы с выбором его типа.(Квадрат и прямоугольник)

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 создают следующие плоские объекты: квадрат, треугольник, прямоугольник, параллелограмм, трапеция, правильный шестиугольник, правильный восьмиугольник (square, triangle, rectangle, parallelogram, trapeze, hexagon, octagon). В базовом классе должны быть объявлены а в производных переопределены виртуальные методы вычисления площади, центра тяжести, а также методы Вращения (Rotate) и Перемещения (Move).

**2.Спецификация класса Interface.h**

**2.1 Спецификация класса Shape**

class Shape {

public:

Point\* arc;

CenterPoint cenPoint;

string Id;

virtual void info() = 0;

virtual ~Shape() {}

static Shape\* createShape(char);

virtual double getArea() = 0;

virtual void Move(double x, double y) = 0;

virtual void Rotate(int degree, double x, double y) = 0;

virtual CenterPoint cal\_Cen\_gravity() = 0;

virtual double getRad() = 0;};

**Класс Shape позволяет выполнять некоторые процедуры над фигурами(передвижение, поворот…)**

**2.2 Спецификация класса Square**

class Square : public Shape {

public:

double side = 0;

void info()

{

cout << "Квадрат" << endl;

}

Square() {

cout << "Укажите уникальный id квадрата" << endl;

cin >> Id;

arc = new Point[4];

cout << "Укажите координаты (x, y) левой нижней грани" << endl;

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

try

{

cout << "Укажите длину стороны" << endl;

cin >> side;

if (side == 0)throw 1;

abs(side);

arc[1].x = arc[0].x;

arc[1].y = arc[0].y + side;

arc[2].x = arc[0].x + side;

arc[2].y = arc[1].y;

arc[3].x = arc[2].x;

arc[3].y = arc[0].y;

cenPoint.x = arc[2].x / 2;

cenPoint.y = arc[2].y / 2;

}

catch (int e)

{

cout << "Ошибка! вы ввели не верную динуну стороны: " << side << endl;

return;

}

}

double getArea()

{

return (side \* side);

}

CenterPoint cal\_Cen\_gravity()

{

cenPoint.x = arc[2].x / 2;

cenPoint.y = arc[2].y / 2;

return cenPoint;

}

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

{

arc[0].x += x;

arc[0].y += y;

arc[1].x += x;

arc[1].y += y;

arc[2].x += x;

arc[2].y += y;

arc[3].x += x;

arc[3].y += y;

cenPoint.x += x;

cenPoint.y += y;

}

void Shape::Rotate(int degree, double x, double y)

{

double a = degree \* PI / 180; //переводим в радианы

arc[0].x = (arc[0].x - x) \* cos(a) - (arc[0].y - y) \* sin(a);

arc[0].y = (arc[0].x - x) \* sin(a) + (arc[0].y - y) \* cos(a);

cout << "x1=" << arc[0].x + x << endl;

cout << "y1=" << arc[0].y + y << endl;

arc[1].x = (arc[1].x - x) \* cos(a) - (arc[1].y - y) \* sin(a);

arc[1].y = (arc[1].x - x) \* sin(a) + (arc[1].y - y) \* cos(a);

cout << "x2=" << arc[1].x + x << endl;

cout << "y2=" << arc[1].y + y << endl;

arc[2].x = (arc[2].x - x) \* cos(a) - (arc[2].y - y) \* sin(a);

arc[2].y = (arc[2].x - x) \* sin(a) + (arc[2].y - y) \* cos(a);

cout << "x3=" << arc[2].x + x << endl;

cout << "y3=" << arc[2].y + y << endl;

arc[3].x = (arc[3].x - x) \* cos(a) - (arc[3].y - y) \* sin(a);

arc[3].y = (arc[3].x - x) \* sin(a) + (arc[3].x - y) \* cos(a);

cout << "x4=" << arc[3].x + x << endl;

cout << "y4=" << arc[3].y + y << endl;

}

double Shape::getRad()

{

return sqrt(pow((side / 2) + (side / 2), 2));

}

~Square()

{

delete[] arc;

}

};

**В классе квадрата мы создаем этот самый квадрат и реализуем все методы Shape(Передвижение, Вращение… )**

**2.3 Спецификация класса Rectangle**

class Rectangle : public Shape

{

public:

double side1 = 0;

double side2 = 0;

void info()

{

cout << "Прямоугольник" << endl;

}

Rectangle()

{

cout << "Укажите уникальный id прямоугольника" << endl;

cin >> Id;

arc = new Point[4];

cout << "Укажите координаты (x, y) левой нижней грани" << endl;

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

try

{

cout << "Укажите ширину" << endl;

cin >> side1;

if (side1 == 0) throw 1;

abs(side1);

cout << "Укажите длину" << endl;

cin >> side2;

if (side2 == 0)throw 2;

abs(side2);

}

catch (int e)

{

if (e == 1)

cout << "Ошибка! вы ввели не верную ширину: " << e << endl;

if (e == 2)

cout << "Ошибка! вы ввели не верную длину: " << e << endl;

return;

}

arc[1].x = arc[0].x;

arc[1].y = arc[0].y + side1;

arc[2].x = arc[0].x + side2;

arc[2].y = arc[1].y;

arc[3].x = arc[2].x;

arc[3].y = arc[0].y;

cenPoint.x = arc[2].x / 2;

cenPoint.y = arc[2].y / 2;

}

double getArea()

{

return side1 \* side2;

}

CenterPoint cal\_Cen\_gravity()

{

cenPoint.x = arc[2].x / 2;

cenPoint.y = arc[2].y / 2;

return cenPoint;

}

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

{

arc[0].x += x;

arc[0].y += y;

arc[1].x += x;

arc[1].y += y;

arc[2].x += x;

arc[2].y += y;

arc[3].x += x;

arc[3].y += y;

cenPoint.x += x;

cenPoint.y += y;

}

void Shape::Rotate(int degree, double x, double y)

{

double a = degree \* PI / 180;

arc[0].x = (arc[0].x - x) \* cos(a) - (arc[0].y - y) \* sin(a);

arc[0].y = (arc[0].x - x) \* sin(a) + (arc[0].y - y) \* cos(a);

cout << "x1=" << arc[0].x + x << endl;

cout << "y1=" << arc[0].y + y << endl;

arc[1].x = (arc[1].x - x) \* cos(a) - (arc[1].y - y) \* sin(a);

arc[1].y = (arc[1].x - x) \* sin(a) + (arc[1].y - y) \* cos(a);

cout << "x2=" << arc[1].x + x << endl;

cout << "y2=" << arc[1].y + y << endl;

arc[2].x = (arc[2].x - x) \* cos(a) - (arc[2].y - y) \* sin(a);

arc[2].y = (arc[2].x - x) \* sin(a) + (arc[2].y - y) \* cos(a);

cout << "x3=" << arc[2].x + x << endl;

cout << "y3=" << arc[2].y + y << endl;

arc[3].x = (arc[3].x - x) \* cos(a) - (arc[3].y - y) \* sin(a);

arc[3].y = (arc[3].x - x) \* sin(a) + (arc[3].x - y) \* cos(a);

cout << "x4=" << arc[3].x + x << endl;

cout << "y4=" << arc[3].y + y << endl;

}

~Rectangle()

{

delete[]arc;

}

double Shape::getRad()

{

return sqrt(pow((side1 / 2) + (side2 / 2), 2));

}

};

//Фабричный метод

Shape\* Shape::createShape(char cd)

{

Shape\* sh = 0;

switch (cd) {

case 'S':

sh = new Square(); break;

case 'R':

sh = new Rectangle(); break;

default: break;

}

return sh;

}

**Всё то же самое проделываем и для прямоугольника в Классе Rectangle**

**2.4Спецификация класса Operat**

class Operat

{

public:

void Compare\_Areas(Shape& a, Shape& b)

{

if (a.getArea() > b.getArea())

cout << "Фигура с Id " << a.Id << " Больше по площади" << endl;

if (a.getArea() < b.getArea())

cout << "Фигура с Id " << b.Id << " Больше по площади" << endl;

if (a.getArea() == b.getArea())

cout << "Фигуры равны по площади" << endl;

}

double Max(double a1, double a2, double a3, double a4)

{

return max(a1, max(a2, max(a3, a4)));

}

double Min(double a1, double a2, double a3, double a4)

{

return min(a1, min(a2, min(a3, a4)));

}

bool IsInclude(Shape& a1, Shape& a2)

{

double minXA1, maxXA1, minXA2, maxXA2;

double minYA1, maxYA1, minYA2, maxYA2;

bool res = false;

minXA1 = Min(a1.arc[0].x, a1.arc[1].x, a1.arc[2].x, a1.arc[3].x);

maxXA1 = Max(a1.arc[0].x, a1.arc[1].x, a1.arc[2].x, a1.arc[3].x);

minYA1 = Min(a1.arc[0].y, a1.arc[1].y, a1.arc[2].y, a1.arc[3].y);

maxYA1 = Max(a1.arc[0].y, a1.arc[1].y, a1.arc[2].y, a1.arc[3].y);

minXA2 = Min(a2.arc[0].x, a2.arc[1].x, a2.arc[2].x, a2.arc[3].x);

maxXA2 = Max(a2.arc[0].x, a2.arc[1].x, a2.arc[2].x, a2.arc[3].x);

minYA2 = Min(a2.arc[0].y, a2.arc[1].y, a2.arc[2].y, a2.arc[3].y);

maxYA2 = Max(a2.arc[0].y, a2.arc[1].y, a2.arc[2].y, a2.arc[3].y);

bool result = false;

if ((minXA2 >= minXA1) && (minYA2 >= minYA1) &&

(maxXA2 <= maxXA1) && (maxYA2 <= maxYA1)

)

result = true;

else

if ((minXA1 >= minXA2) && (minYA1 >= minYA2) &&

(maxXA1 <= maxXA2) && (maxYA1 <= maxYA2)

)

result = true;

return result;

}

bool IsIntersect(Shape& a, Shape& b)

{

double dist;

double c1, c2;

c1 = fabs(a.cenPoint.x - b.cenPoint.x);

c2 = fabs(a.cenPoint.y - b.cenPoint.y);

dist = sqrt(c1 \* c1 + c2 \* c2);

return (dist <= (a.getRad() + b.getRad()));

}

};

**Класс Operate даёт нам возможность сравнивать наши фигуры как между собой так и их расположение на плоскости(пересекаются ли они)**

**3.Диаграмма классов**

**4.Листинг программы**

#include<iostream>

#include<cmath>

#include <string>

const double PI = 3.141592653589793;

#define \_USE\_MATH\_DEFINES

using namespace::std;

struct Point { double x, y; };

struct CenterPoint { double x, y; };

**Создаём 2 структуры для Точки и Центральной точки**

int main()

{

setlocale(LC\_ALL, "Russian");

Shape\* p1 = 0;

Operat oper;

while (true)

{

char figur;

cout << "Введите индетификатор фигуры, которую хотите создать (S - квадрат R - прямоугольник)" << endl;

cin >> figur;

if (figur == 'S' or figur == 'R')

{

p1 = Shape::createShape(figur);

if (p1->getArea() != 0)

{

cout << "x1 = " << p1->arc[0].x << " y1 = " << p1->arc[0].y << endl;

cout << "x2 = " << p1->arc[1].x << " y2 = " << p1->arc[1].y << endl;

cout << "x3 = " << p1->arc[2].x << " y3 = " << p1->arc[2].y << endl;

cout << "x4 = " << p1->arc[3].x << " y4 = " << p1->arc[3].y << endl;

break;

}

else

{

cout << "Вы ввели не верные параметры при создании фигуры" << endl;

}

}

else

cout << "Вы ввели не существующую фигуру" << endl;

}

Shape\* p2 = 0;

while (true)

{

char figur;

cout << "Введите индетификатор фигуры, которую хотите создать (S - квадрат R - прямоугольник)" << endl;

cin >> figur;

if (figur == 'S' or figur == 'R')

{

p2 = Shape::createShape(figur);

if (p2->getArea() != 0)

{

cout << "x1 = " << p2->arc[0].x << " y1 = " << p2->arc[0].y << endl;

cout << "x2 = " << p2->arc[1].x << " y2 = " << p2->arc[1].y << endl;

cout << "x3 = " << p2->arc[2].x << " y3 = " << p2->arc[2].y << endl;

cout << "x4 = " << p2->arc[3].x << " y4 = " << p2->arc[3].y << endl;

break;

}

else

{

cout << "Вы ввели не верные параметры при создании фигуры" << endl;

}

}

else

cout << "Вы ввели не существующую фигуру" << endl;

}

while (true)

{

int i = 0;

cout << "1.Перемещение" << endl;

cout << "2.Вращение" << endl;

cout << "3.Сравнить по площади" << endl;

cout << "4.Пересекаются ли фигуры" << endl;

cout << "5.Содержит ли одна фигура другую" << endl;

cout << "6.выйти" << endl;

cin >> i;

switch (i)

{

case(1):

{ string a;

cout << "Введите индетификатор фигуры" << endl;

cin >> a;

if (a == (p1->Id))

{

double x, y;

cout << "Введите координаты смещения (x, y)" << endl;

cin >> x >> y;

p1->Move(x, y);

cout << "Координаты после смещения" << endl;

cout << "x1 = " << p1->arc[0].x << " y1 = " << p1->arc[0].y << endl;

cout << "x2 = " << p1->arc[1].x << " y2 = " << p1->arc[1].y << endl;

cout << "x3 = " << p1->arc[2].x << " y3 = " << p1->arc[2].y << endl;

cout << "x4 = " << p1->arc[3].x << " y4 = " << p1->arc[3].y << endl;

}

if (a == (p2->Id))

{

double x, y;

cout << "Введите координаты смещения (x, y)" << endl;

cin >> x >> y;

p2->Move(x, y);

cout << "Координаты после смещения" << endl;

cout << "x1 = " << p2->arc[0].x << " y1 = " << p2->arc[0].y << endl;

cout << "x2 = " << p2->arc[1].x << " y2 = " << p2->arc[1].y << endl;

cout << "x3 = " << p2->arc[2].x << " y3 = " << p2->arc[2].y << endl;

cout << "x4 = " << p2->arc[3].x << " y4 = " << p2->arc[3].y << endl;

}

else

cout << "Не существует фигуры с таким идентификатором" << endl;

break;

}

case(2):

{ string a;

cout << "Введите индетификатор фигуры" << endl;

cin >> a;

if (a == p1->Id)

{

double rt, x, y;

cout << "Введите градус поворота и координаты точки вращения (градус , x , y)" << endl;

cin >> rt >> x >> y;

p1->Rotate(rt, x, y);

cout << "Координаты после вращения" << endl;

cout << "x1 = " << p1->arc[0].x << " y1 = " << p1->arc[0].y << endl;

cout << "x2 = " << p1->arc[1].x << " y2 = " << p1->arc[1].y << endl;

cout << "x3 = " << p1->arc[2].x << " y3 = " << p1->arc[2].y << endl;

cout << "x4 = " << p1->arc[3].x << " y4 = " << p1->arc[3].y << endl;

}

if (a == p2->Id)

{

double rt, x, y;

cout << "Введите градус поворота и координаты точки вращения (градус , x , y)" << endl;

cin >> rt >> x >> y;

p1->Rotate(rt, x, y);;

cout << "Координаты после вращения" << endl;

cout << "x1 = " << p2->arc[0].x << " y1 = " << p2->arc[0].y << endl;

cout << "x2 = " << p2->arc[1].x << " y2 = " << p2->arc[1].y << endl;

cout << "x3 = " << p2->arc[2].x << " y3 = " << p2->arc[2].y << endl;

cout << "x4 = " << p2->arc[3].x << " y4 = " << p2->arc[3].y << endl;

}

}

case(3):

{

oper.Compare\_Areas(\*p1, \*p2);

break;

}

case(4):

{

if (oper.IsIntersect(\*p1, \*p2))

cout << "Да" << endl;

else

cout << "Нет" << endl;

break;

}

case(5):

{

if (oper.IsInclude(\*p1, \*p2))

cout << "Да" << endl;

else

cout << "Нет" << endl;

break;

}

case(6):

{

return 0;

}

}

}

}

**Создаём 2 фигуры, вводим их координаты, выбираем действие которое хотим совершить с ней/ними.**

**Case 1 – Перемещение**

**Case 2 – Вращение**

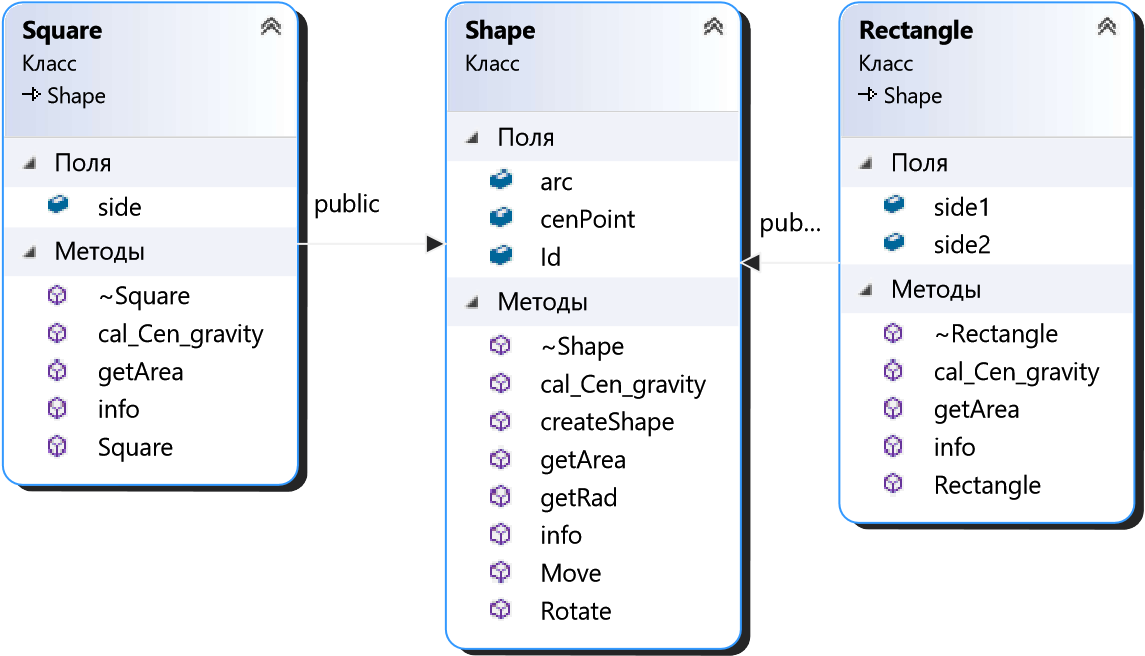
**Case 3 – Сравнение по площади**

**Case 4 – Пересечение фигур**

**Case 5 – Нахождение одной фигуры в другой**

**Case 6 – Выход**

**Диаграмма классов**





Operat

Класс



Методы



Compare\_Areas



IsInclude



IsIntersect



Max



Min