**МИНОБРНАУКИ РОССИИ**

**ФЕДЕРАЛЬАНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения ЭВМ**

**Отчёт по лабораторной работе №3**

по курсу «Объектно-ориентированное программирование»

на тему: **«Динамический выбор типа объектов»**

Выполнил:  
студент гр. КТбо1-6

Бабанский И.П.

Проверил:

доцент МОП ЭВМ, к. т. н.  
Лутай В.Н.

Вариант №1

Постановка задачи

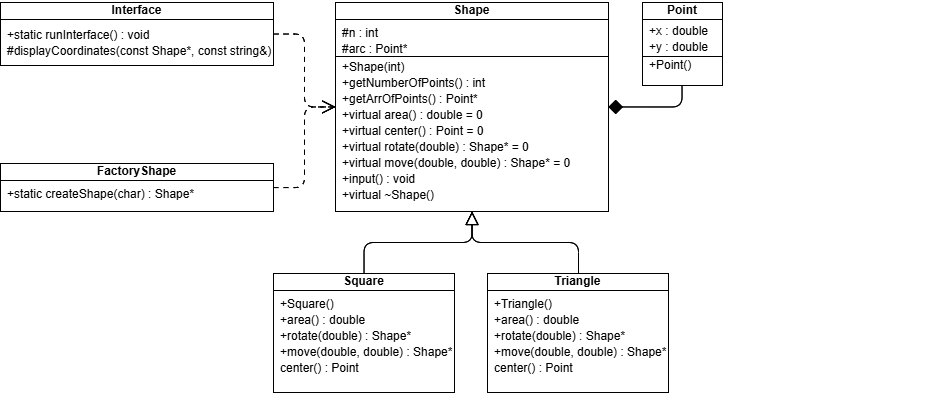
Создать программу, создающие объекты двух классов: Triangle и Square, производные от класса Shape. В базовом классе должны быть объявлены, а в производных переопределены виртуальные методы вычисления площади, центра тяжести, а также методы Вращения (Rotate) и Перемещения (Move). необходимо создать операции над объектами 2˗х классов: сравнить два объекта по площади – Compare, проверить факт пересечения объектов – isIntersect, проверить факт включения одного объекта в другой – isInclude.

Спецификация класса

Спецификация класса в лабораторной работе размещена в заголовочном файле **Shape.hpp**.

В файле объявлен класс Point с полями double x и y для хранения значений координат. Объявлен Shape, являющийся родительским классом для Triangle и Square, с полями n - количеством точек фигуры – и arc – массивом точек типа Point. В Shape есть чистые виртуальные функции: area – для вычисления площади, center – для вычисления центра тяжести фигуры, rotate – для вращения, и move – для перемещения, и невиртуальные функции–геттеры getNumberOfPoints и getArrOfPoints, сделанные таковыми, т.к. их код в производных классах никак не отличался бы. Так же есть метод ввода, конструктор – тоже одни для всех производных классов – и деструктор. В Triangle и Square переопределяются все виртуальные функции родительского класса. Объявлен FactoryShape, реализующий «Фабричный метод» проектирования программы. Его единственный метод – createShape, возвращающий ссылку на новый запрошенный объект. Объявлен Operators – класс со статическими методами-операциями, которые можно проводить с фигурами. В нем, помимо необходимых, объявлены вспомогательные методы, необходимые для реализации операций. Так, left, определяющий положение точки относительно прямой, необходим для isIntersect, определяющего, пересекаются ли фигуры, и isInclude, определяющего, включаются ли фигуры друг в друга, а segmentIntersection, определяющий, пересекаются ли отрезки, и onSegment, проверяющий принадлежность точки отрезку, помогают только isIntersect. Последним объявленным в файле класс является Interface, служащий интерфейсом между консолью и программой. Помимо void-функции run, запускающей циклы опросов пользователя, в нем объявлен вспомогательный метод displayCoordinates для отображения координат фигуры.

Диаграмма классов:



Листинг программы:

**Shape.hpp:**

#ifndef SHAPE\_HPP

#define SHAPE\_HPP

#include <iostream>

#include <cmath>

const double M\_PI = 3.14159265358979323846;

class Point {

public:

Point();

double x, y;

};

class Shape {

protected:

Point\* arc = nullptr;

int n = 0;

public:

Shape(int numPoints);

int getNumberOfPoints() const;

Point\* getArrOfPoints() const;

virtual double area() const = 0;

virtual Point center() const = 0;

virtual Shape\* rotate(double angle) const = 0;

virtual Shape\* move(double dx, double dy) const = 0;

void input();

virtual ~Shape();

};

class Triangle : public Shape {

public:

Triangle();

double area() const override;

Shape\* rotate(double angle) const override;

Shape\* move(double dx, double dy) const override;

Point center() const override;

};

class Square : public Shape {

public:

Square();

double area() const override;

Shape\* rotate(double angle) const override;

Shape\* move(double dx, double dy) const override;

Point center() const override;

};

class FactoryShape {

public:

static Shape\* createShape(char type);

};

class Operations {

private:

static int left(Point& cur, Point& frst, Point& scnd);

static bool segmentIntersection(Point& frstA, Point& frstB, Point& scndA, Point& scndB);

static bool onSegment(Point& cur, Point& frst, Point& scnd);

public:

static bool isIntersect(Shape\* first, Shape\* second);

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

static int isInclude(const Shape\* shape1, const Shape\* shape2);

};

class Interface {

public:

static void runInterface();

protected:

static void displayCoordinates(const Shape\* shape, const string& shapeName); // Добавлен спецификатор void

};

#endif // SHAPE\_HPP

**Shape.cpp:**

#include "Shape.hpp"

using namespace std;

Point::Point() : x(0.0), y(0.0) {}

Shape::Shape(int numPoints) : n(numPoints), arc(new Point[n]) {}

Shape::~Shape() { delete[] arc; }

int Shape::getNumberOfPoints() const { return n; };

Point\* Shape::getArrOfPoints() const { return arc; }

void Shape::input() {

for (int i = 0; i < n; i++){

cout << i+1 << ": ";

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

}

}

Triangle::Triangle() : Shape(3) {}

Shape\* Triangle::rotate(double angle) const {

Point c = center();

double rad = angle \* M\_PI / 180.0;

Triangle\* newTriangle = new Triangle();

for (int i = 0; i < n; i++) {

newTriangle->arc[i].x = c.x + (arc[i].x - c.x) \* cos(rad) - (arc[i].y - c.y) \* sin(rad);

newTriangle->arc[i].y = c.y + (arc[i].x - c.x) \* sin(rad) + (arc[i].y - c.y) \* cos(rad);

}

return newTriangle;

};

Shape\* Triangle::move(double dx, double dy) const {

Triangle\* newTriangle = new Triangle();

for (int i = 0; i < n; i++) {

newTriangle->arc[i].x = arc[i].x + dx;

newTriangle->arc[i].y = arc[i].y + dy;

}

return newTriangle;

}

double Triangle::area() const {

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

arc[1].x \* (arc[2].y - arc[0].y) +

arc[2].x \* (arc[0].y - arc[1].y));

}

Point Triangle::center() const {

Point c;

c.x = (arc[0].x + arc[1].x + arc[2].x) / 3;

c.y = (arc[0].y + arc[1].y + arc[2].y) / 3;

return c;

}

Square::Square() : Shape(4) {}

Shape\* Square::rotate(double angle) const {

angle = fmod(angle, 360.0);

if (angle < 0) {

angle += 360.0;

}

Point c = center();

double rad = angle \* M\_PI / 180.0;

Square\* newSquare = new Square();

for (int i = 0; i < n; i++) {

newSquare->arc[i].x = c.x + (arc[i].x - c.x) \* cos(rad) - (arc[i].y - c.y) \* sin(rad);

newSquare->arc[i].y = c.y + (arc[i].x - c.x) \* sin(rad) + (arc[i].y - c.y) \* cos(rad);

}

return newSquare;

}

Shape\* Square::move(double dx, double dy) const {

Square\* newSquare = new Square();

for (int i = 0; i < n; i++) {

newSquare->arc[i].x = arc[i].x + dx;

newSquare->arc[i].y = arc[i].y + dy;

}

return newSquare;

}

double Square::area() const {

double width = abs(arc[0].x - arc[1].x);

double height = abs(arc[0].y - arc[2].y);

return width \* height;

}

Point Square::center() const {

Point c;

c.x = (arc[0].x + arc[2].x) / 2;

c.y = (arc[0].y + arc[2].y) / 2;

return c;

}

int Operations::left(Point& cur, Point& frst, Point& scnd) {

double d = (scnd.x - frst.x) \* (cur.y - frst.y) - (scnd.y - frst.y) \* (cur.x - frst.x);

return (d > 0 ? 1 : (d == 0 ? 0 : -1));

}

bool Operations::segmentIntersection(Point& frstA, Point& frstB, Point& scndA, Point& scndB){

double o1 = left(frstA, scndA, scndB);

double o2 = left(frstB, scndA, scndB);

double o3 = left(scndA, frstA, frstB);

double o4 = left(scndB, frstA, frstB);

//Отрезки пересекаются, если точки одного находятся по разные стороны другого

if (o1 != o2 && o3 != o4) return true;

//Если точка лежит на прямой отрезка, то нужно проверить, принадлежит ли она ему

if (o1 == 0 && onSegment(frstA, scndA, scndB)) return true;

if (o2 == 0 && onSegment(frstB, scndA, scndB)) return true;

if (o3 == 0 && onSegment(scndA, frstA, frstB)) return true;

if (o4 == 0 && onSegment(scndB, frstA, frstB)) return true;

return false;

};

bool Operations::onSegment(Point& cur, Point& frst, Point& scnd) {

return cur.x <= max(frst.x, scnd.x) && cur.x >= min(frst.x, scnd.x) &&

cur.y <= max(frst.y, scnd.y) && cur.y >= min(frst.y, scnd.y);

}

bool Operations::isIntersect(Shape\* first, Shape\* second) {

Point\* firstPoints = first->getArrOfPoints();

Point\* secondPoints = second->getArrOfPoints();

int n1 = first->getNumberOfPoints();

int n2 = second->getNumberOfPoints();

for (int i = 0; i < n1; i++) {

for (int j = 0; j < n2; j++) {

if (segmentIntersection(firstPoints[i], firstPoints[(i + 1) % n1],

secondPoints[j], secondPoints[(j + 1) % n2])) {

return true;

}

}

}

return false;

}

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

double area1 = shape1->area();

double area2 = shape2->area();

if (area1 > area2)

return 1;

else if (area1 < area2)

return -1;

else

return 0;

}

int Operations::isInclude(const Shape\* shape1, const Shape\* shape2) {

Point\* arr1 = shape1->getArrOfPoints();

Point\* arr2 = shape2->getArrOfPoints();

int n1 = shape1->getNumberOfPoints();

int n2 = shape2->getNumberOfPoints();

int tmp;

bool shape1\_in\_shape2 = true, shape2\_in\_shape1 = true, Rflag, Lflag;

for (int i = 0; i < n1 && shape2\_in\_shape1; i++) {

Rflag = false, Lflag = false;

for (int j = 0; j < n2; j++) {

Point cur = arr1[i];

Point frst = arr2[j];

Point scnd = arr2[(j + 1) % n2];

tmp = left(cur, frst, scnd);

if (tmp == 1)

Lflag = true;

else if (tmp == -1)

Rflag = true;

}

if(Lflag && Rflag)

shape1\_in\_shape2 = false;

}

if(!shape1\_in\_shape2){

for (int i = 0; i < n2 && shape2\_in\_shape1; i++) {

Rflag = false, Lflag = false;

for (int j = 0; j < n1; j++) {

Point cur = arr2[i];

Point frst = arr1[j];

Point scnd = arr1[(j + 1) % n1];

tmp = left(cur, frst, scnd);

if (tmp == 1)

Lflag = true;

else if (tmp == -1)

Rflag = true;

}

if(Lflag && Rflag)

shape2\_in\_shape1 = false;

}

}

if (shape1\_in\_shape2) return 1; // Все точки shape1 внутри shape2

if (shape2\_in\_shape1) return -1; // Все точки shape2 внутри shape1

return 0; // Неполное включение или пересечение

}

Shape\* FactoryShape::createShape(char type) {

switch (type) {

case 'T':

return new Triangle;

case 'S':

return new Square;

default:

return nullptr;

}

}

void Interface::runInterface() {

Shape\* shape1 = nullptr;

Shape\* shape2 = nullptr;

system("chcp 1251");

while (true) {

cout << "Выберите фигуру для создания:" << endl;

cout << "1. Треугольник (T)" << endl;

cout << "2. Квадрат (S)" << endl;

cout << "0. Выход" << endl;

char choice;

cin >> choice;

if (choice == '0') {

break;

}

shape1 = FactoryShape::createShape(choice);

if (shape1 == nullptr) {

cout << "Некорректный код выбора." << endl;

continue;

}

cout << "Введите координаты фигуры (вписывайте координаты точек в порядке обхожа фигуры в любом направлении):" << endl;

shape1->input();

cout << "Создана фигура: " << typeid(\*shape1).name() << endl;

cout << "Выберите вторую фигуру для сравнения:" << endl;

cin >> choice;

shape2 = FactoryShape::createShape(choice);

if (shape2 == nullptr) {

cout << "Некорректный код выбора." << endl;

delete shape1;

continue;

}

cout << "Введите координаты фигуры (вписывайте координаты точек в порядке обхожа фигуры в любом направлении):" << endl;

shape2->input();

cout << "Создана фигура: " << typeid(\*shape2).name() << endl;

while (true) {

cout << "Выберите операцию:" << endl;

cout << "1. Сравнить площади" << endl;

cout << "2. Проверить включение" << endl;

cout << "3. Проверить пересечение" << endl;

cout << "4. Получить центры фигур" << endl;

cout << "5. Повернуть фигуру 1" << endl;

cout << "6. Повернуть фигуру 2" << endl;

cout << "7. Сдвинуть фигуру 1" << endl;

cout << "8. Сдвинуть фигуру 2" << endl;

cout << "9. Отобразить координаты фигур" << endl;

cout << "0. Вернуться к выбору фигур" << endl;

int operation;

cin >> operation;

switch (operation) {

case 1: {

double area1 = shape1->area();

double area2 = shape2->area();

int result = Operations::Compare(shape1, shape2);

if (result == 1) {

cout << "Фигура 1 больше по площади: " << area1 << " > " << area2 << "" << endl;

} else if (result == -1){

cout << "Фигура 2 больше по площади: " << area1 << " < " << area2 << endl;

} else {

cout << "Фигуры равны по площади: " << area1 << "" << endl;

}

break;

}

case 2: {

int result = Operations::isInclude(shape1, shape2);

if (result == 1) {

cout << "Фигура 1 включена в фигуру 2." << endl;

} else if (result == -1) {

cout << "Фигура 2 включена в фигуру 1." << endl;

} else {

cout << "Ни одна фигура не включает другую." << endl;

}

break;

}

case 3: {

if (Operations::isIntersect(shape1, shape2)) {

cout << "Фигуры пересекаются." << endl;

} else {

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

}

break;

}

case 4: {

Point center1 = shape1->center();

Point center2 = shape2->center();

cout << "Центр фигуры 1: (" << center1.x << ", " << center1.y << ")" << endl;

cout << "Центр фигуры 2: (" << center2.x << ", " << center2.y << ")" << endl;

break;

}

case 5: {

double angle;

cout << "Введите угол поворота (положительное число, означающее поворот по часовой стрелке в градусах): ";

cin >> angle;

Shape\* newShape1 = shape1->rotate(angle);

delete shape1; // Освобождаем память старой фигуры

shape1 = newShape1; // Заменяем указатель на новый объект

cout << "Фигура 1 повернута на " << angle << " градусов." << endl;

break;

}

case 6: {

double angle;

cout << "Введите угол поворота (положительное число, означающее поворот по часовой стрелке в градусах): ";

cin >> angle;

Shape\* newShape2 = shape2->rotate(angle);

delete shape2; // Освобождаем память старой фигуры

shape2 = newShape2; // Заменяем указатель на новый объект

cout << "Фигура 2 повернута на " << angle << " градусов." << endl;

break;

}

case 7: {

double dx, dy;

cout << "Введите смещение по X и Y: ";

cin >> dx >> dy;

Shape\* newShape1 = shape1->move(dx, dy);

delete shape1; // Освобождаем память старой фигуры

shape1 = newShape1; // Заменяем указатель на новый объект

cout << "Фигура 1 сдвинута на (" << dx << ", " << dy << ")." << endl;

break;

}

case 8: {

double dx, dy;

cout << "Введите смещение по X и Y: ";

cin >> dx >> dy;

Shape\* newShape2 = shape2->move(dx, dy);

delete shape2; // Освобождаем память старой фигуры

shape2 = newShape2; // Заменяем указатель на новый объект

cout << "Фигура 2 сдвинута на (" << dx << ", " << dy << ")." << endl;

break;

}

case 9: {

displayCoordinates(shape1, "фигуры 1");

displayCoordinates(shape2, "фигуры 2");

break;

}

case 0:

goto end\_operations; // Возврат к выбору фигуры

default:

cout << "Некорректный выбор операции." << endl;

break;

}

}

end\_operations:

delete shape1;

delete shape2;

}

}

void Interface::displayCoordinates(const Shape\* shape, const string& shapeName) {

cout << "Координаты " << shapeName << ":" << endl;

int n = shape->getNumberOfPoints();

Point\* points = shape->getArrOfPoints();

for (int i = 0; i < n; ++i) {

cout << "Точка " << i + 1 << ": (" << points[i].x << ", " << points[i].y << ")" << endl;

}

}

**Main.cpp**

#include <iostream>

#include "Shape.hpp"

using namespace std;

int main()

{

Interface::runInterface();

return 0;

}