МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Лабораторная работа № 3**

по курсу «Объектно-ориентированное программирование»

**«Динамический выбор типа объектов»**

**1 ВАРИАНТ**

Выполнили:

студенты гр. КТбо2-1

Неприн М.А.

Проверил:

Тарасов С. А.

**Таганрог 2020**

1. **Вариант задания №10**

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).

Особенностью этой программы является использование статического метода createShape, который можно вызывать без создания объекта соответствующего класса. Этот метод ( он может быть и нестатическим) называется фабричным из-за того, что он, не будучи конструктором, создает объект класса. При этом точка создания объекта отделена от соответствующего класса. 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. Эти методы также можно делать статическими.

1. **Спецификация классов**

class Shape

{

public:

Shape(Point& point, double\* edgeslength) : point(point), edgeslength(edgeslength) {};

void virtual Move(Point &point);

void virtual Rotate(int rotation);

double virtual getArea() const { return area; }

Point virtual const &getCenterOfMass() const { return point; }

double virtual getPerimeter() const { return perimeter; }

int virtual getRotation() const { return rotation; }

Point virtual const &getPoint() const { return point; }

int virtual getEdgesCount() const { return edgescount; }

double virtual \*getEdgesLength() const { return edgeslength; }

std::string virtual getName() const = 0;

double virtual getMaxRadius() const { return maxradius; }

protected:

void virtual countPerimeter() = 0;

void virtual countArea() = 0;

void virtual countMaxRadius() = 0;

const int edgescount = 0;

double\* edgeslength = nullptr;

double maxradius = 0;

double area = 0;

double perimeter = 0;

int rotation = 90;

Point point;

};

class Square : public Shape

{

public:

Square(Point& point, double\* edgeslength);

std::string getName() const { return "Square"; }

private:

const int edgescount = 4;

void countPerimeter() override;

void countArea() override;

void countMaxRadius() override;

};

class Triangle : public Shape

{

public:

Triangle(Point& point, double\* edgeslength);

std::string getName() const { return "Triangle"; }

private:

const int edgescount = 3;

void countPerimeter() override;

void countArea() override;

void countMaxRadius() override;

};

class Operations

{

public:

static int compare(Shape& shape1, Shape& shape2);

static bool isIncluded(Shape& shape1, Shape& shape2);

static bool isIntersected(Shape& shape1, Shape& shape2);

static double getDistance(Point& point1, Point& point2);

};

class ShapeFactory

{

public:

static Shape\* CreateShape(char cd, Point& point, double\* edgeslength);

};

class ConsoleInterface

{

public:

ConsoleInterface() {};

void run();

void printMenuText();

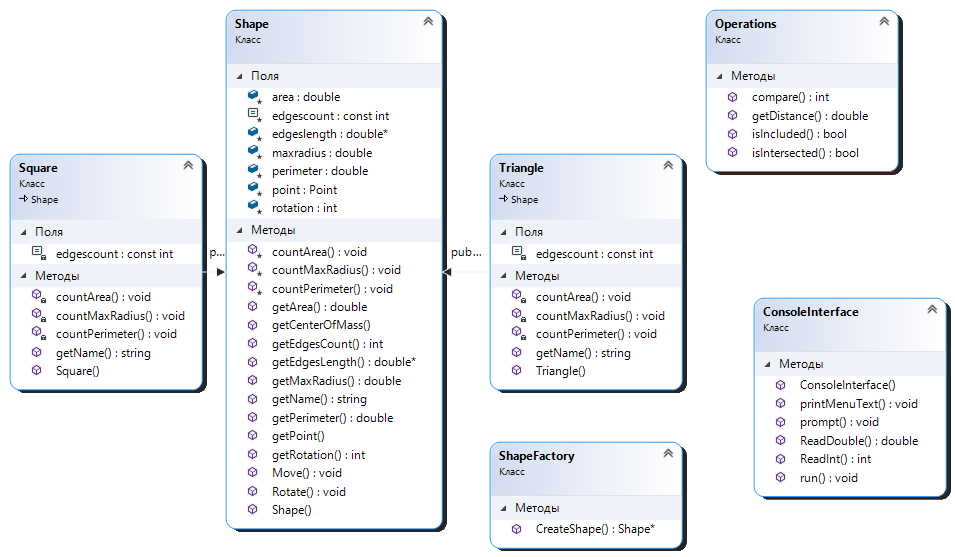
int ReadInt();

double ReadDouble();

void prompt();

};

1. **Диаграммы классов**



1. **Листинг программы**

**main.cpp**

/\*

// ЮФУ ИКТИБ МОП ЭВМ

// Неприн Михаил Андреевич, КТбо2-1

// Лабораторная работ №3, Вариант №1

// 14.11.2020

\*/

#include "ConsoleInterface.h"

int main()

{

ConsoleInterface CI;

CI.run();

return 0;

}

**Shape.h**

#pragma once

#include <string>

struct Point { double x, y; };

class Shape

{

public:

Shape(Point& point, double\* edgeslength) : point(point), edgeslength(edgeslength) {};

void virtual Move(Point &point);

void virtual Rotate(int rotation);

double virtual getArea() const { return area; }

Point virtual const &getCenterOfMass() const { return point; }

double virtual getPerimeter() const { return perimeter; }

int virtual getRotation() const { return rotation; }

Point virtual const &getPoint() const { return point; }

int virtual getEdgesCount() const { return edgescount; }

double virtual \*getEdgesLength() const { return edgeslength; }

std::string virtual getName() const = 0;

double virtual getMaxRadius() const { return maxradius; }

protected:

void virtual countPerimeter() = 0;

void virtual countArea() = 0;

void virtual countMaxRadius() = 0;

const int edgescount = 0;

double\* edgeslength = nullptr;

double maxradius = 0;

double area = 0;

double perimeter = 0;

int rotation = 90;

Point point;

};

**Shape.cpp**

#include "Shape.h"

void Shape::Move(Point &point)

{

this->point = point;

}

void Shape::Rotate(int rotation)

{

this->rotation = rotation;

}

**Square.h**

#pragma once

#include "Shape.h"

class Square : public Shape

{

public:

Square(Point& point, double\* edgeslength);

std::string getName() const { return "Square"; }

private:

const int edgescount = 4;

void countPerimeter() override;

void countArea() override;

void countMaxRadius() override;

};

**Square.cpp**

#include "Square.h"

#include <cmath>

Square::Square(Point& point, double\* edgeslength) : Shape(point, edgeslength)

{

countPerimeter();

countArea();

countMaxRadius();

return;

}

void Square::countArea()

{

area = edgeslength[0]\*2;

}

void Square::countMaxRadius()

{

maxradius = edgeslength[0] / sqrt(2);

}

void Square::countPerimeter()

{

perimeter = edgeslength[0] \* 4;

}

**Triangle.h**

#pragma once

#include "Shape.h"

class Triangle : public Shape

{

public:

Triangle(Point& point, double\* edgeslength);

std::string getName() const { return "Triangle"; }

private:

const int edgescount = 3;

void countPerimeter() override;

void countArea() override;

void countMaxRadius() override;

};

**Triangle.cpp**

#include "Triangle.h"

#include <cmath>

Triangle::Triangle(Point& point, double\* edgeslength) : Shape(point, edgeslength)

{

countPerimeter();

countArea();

countMaxRadius();

return;

}

void Triangle::countArea()

{

double p = perimeter / 2;

area = sqrt(p \* (p - edgeslength[0]) \* (p - edgeslength[1]) \* (p - edgeslength[2]));

}

void Triangle::countPerimeter()

{

perimeter = edgeslength[0] + edgeslength[1] + edgeslength[2];

}

void Triangle::countMaxRadius()

{

double p = perimeter / 2;

maxradius = (edgeslength[0] \* edgeslength[1] \* edgeslength[2]) / (4 \* sqrt(p \* (p - edgeslength[0]) \* (p - edgeslength[1]) \* (p - edgeslength[2])));

}

**Operations.h**

#pragma once

#include "Shape.h"

class Operations

{

public:

static int compare(Shape& shape1, Shape& shape2);

static bool isIncluded(Shape& shape1, Shape& shape2);

static bool isIntersected(Shape& shape1, Shape& shape2);

static double getDistance(Point& point1, Point& point2);

};

**Operations.cpp**

#include "Operations.h"

int Operations::compare(Shape& shape1, Shape& shape2)

{

if (shape1.getArea() > shape2.getArea())

return 1;

else if (shape1.getArea() < shape2.getArea())

return -1;

else

return 0;

}

bool Operations::isIncluded(Shape& shape1, Shape& shape2)

{

double radius1 = shape1.getMaxRadius();

double radius2 = shape2.getMaxRadius();

Point point1 = shape1.getPoint();

Point point2 = shape2.getPoint();

double dist = getDistance(point1, point2);

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

{

return true;

}

return false;

}

bool Operations::isIntersected(Shape& shape1, Shape& shape2)

{

double radius1 = shape1.getMaxRadius();

double radius2 = shape2.getMaxRadius();

Point point1 = shape1.getPoint();

Point point2 = shape2.getPoint();

double dist = getDistance(point1, point2);

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

{

return false;

}

return true;

}

double Operations::getDistance(Point& point1, Point& point2)

{

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

}

**ShapeFactory.h**

#pragma once

#include "Shape.h"

#include "Square.h"

#include "Triangle.h"

class ShapeFactory

{

public:

static Shape\* CreateShape(char cd, Point& point, double\* edgeslength);

};

**ShapeFactory.cpp**

#include "ShapeFactory.h"

#include "Square.h"

#include "Triangle.h"

Shape\* ShapeFactory::CreateShape(char cd, Point &point, double \*edgeslength)

{

Shape\* sh = nullptr;

switch (cd)

{

case 'S': sh = new Square(point, edgeslength); break;

case 'T': sh = new Triangle(point, edgeslength); break;

}

return sh;

}

**ConsoleInterface.h**

#pragma once

class ConsoleInterface

{

public:

ConsoleInterface() {};

void run();

void printMenuText();

int ReadInt();

double ReadDouble();

void prompt();

};

**ConsoleInterface.cpp**

#include "ConsoleInterface.h"

#include "ShapeFactory.h"

#include "Operations.h"

#include <iostream>

enum commands { \_EXIT, \_CREATESHAPE, \_ISINTERSECTED, \_ISINCLUDES, \_SHOWSHAPES, \_PRINTMENUTEXT };

void ConsoleInterface::run()

{

bool running = true;

int shapes\_max = 10;

Shape\*\* shapes = new Shape \* [shapes\_max] {nullptr};

int sh\_counter = 0;

printMenuText();

while (running)

{

int input;

input = ReadInt();

switch (input) {

case \_EXIT:

running = false;

break;

case \_CREATESHAPE: {

char cd;

bool ok = false;

while (!ok)

{

std::cout << "Choose a shape to create ('S' - Square, 'T' - Triangle): ";

std::cin >> cd;

cd = toupper(cd);

if (cd == '0')

{

break;

}

if (cd != 'S' && cd != 'T') {

ok = false;

}

else {

ok = true;

}

Point point;

std::cout << " Write x: " << std::endl;

point.x = ReadDouble();

std::cout << " Write y: " << std::endl;

point.y = ReadDouble();

double sides[3]{NULL};

std::cout << "Write edges " << std::endl;

std::cout << "Write A: " << std::endl;

sides[0] = ReadDouble();

if (cd == 'T')

{

std::cout << "Write B: " << std::endl;

sides[1] = ReadDouble();

std::cout << "Write C: " << std::endl;

sides[2] = ReadDouble();

}

Shape \*nsh = ShapeFactory::CreateShape(cd, point, sides);

if (nsh != nullptr)

{

shapes[sh\_counter++] = nsh;

std::cout << nsh->getName() << " Created";

}

else

{

std::cout << "Shape not created";

}

std::cout << std::endl;

}

break;

}

case \_ISINTERSECTED: {

int f, s;

std::cout << "Write first shape: " << std::endl;

f = ReadInt();

if (f == 0)

break;

std::cout << "Write second shape: " << std::endl;

s = ReadInt();

if (f == 0)

break;

if ((shapes[f-1] != nullptr) && (shapes[s-1] != nullptr)) {

if (Operations::isIntersected(\*shapes[f - 1], \*shapes[s - 1]))

{

std::cout << "They are intersect" << std::endl;

}

else

{

std::cout << "They are not intersect" << std::endl;

}

}

else

{

std::cout << "Invalid shape" << std::endl;

}

printMenuText();

break;

}

case \_ISINCLUDES: {

int f, s;

std::cout << "Write first shape: " << std::endl;

f = ReadInt();

if (f == 0)

break;

std::cout << "Write second shape: " << std::endl;

s = ReadInt();

if (f == 0)

break;

if (Operations::isIncluded(\*shapes[f - 1], \*shapes[s - 1]))

{

std::cout << shapes[f-1]->getName() << " include " << shapes[s - 1]->getName() << std::endl;

}

else

{

std::cout << shapes[f - 1]->getName() << " not include " << shapes[s - 1]->getName() << std::endl;

}

printMenuText();

break;

}

case \_SHOWSHAPES: {

for (int i = 0; (i < shapes\_max); ++i)

{

if (shapes[i] != nullptr)

{

std::cout << i + 1 << ". " << shapes[i]->getName() << std::endl;

}

}

break;

}

case \_PRINTMENUTEXT: {

printMenuText();

break;

}

} // while (running)

}

}

void ConsoleInterface::printMenuText()

{

std::cout << \_CREATESHAPE << " - Create Shape\n"

<< \_ISINTERSECTED << " - To check if shapes are intersect\n"

<< \_ISINCLUDES << " - To check if first shape includes second\n"

<< \_SHOWSHAPES << " - To print all shapes\n"

<< \_PRINTMENUTEXT << " - To print this text\n"

<< \_EXIT << " - To exit\n";

return;

}

void ConsoleInterface::prompt()

{

std::cout << "$ ";

return;

}

int ConsoleInterface::ReadInt()

{

using std::cin;

int a = 0;

prompt();

cin >> a;

if (cin.fail())

{

do

{

prompt();

cin.clear();

cin.ignore(INT64\_MAX, '\n');

cin >> a;

} while (cin.fail());

}

return a;

}

double ConsoleInterface::ReadDouble()

{

using std::cin;

double a = 0;

prompt();

cin >> a;

if (cin.fail())

{

do

{

prompt();

cin.clear();

cin.ignore(INT64\_MAX, '\n');

cin >> a;

} while (cin.fail());

}

return a;

}