МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 1

по дисциплине «Объектно-ориентированное программирование»

Направление подготовки: 230105 - «Программное обеспечение вычислительной техники и автоматизированных систем»

Выполнил слушатель: Сокотущенко Иван Игоревич
Вариант: 2
Дата сдачи:
Преподаватель: Силов Я. В.

Новосибирск, 2017г.

1. Задание

Изучить основные понятия и функции графического интерфейса операционной системы Microsoft Windows. По предложенному преподавателем варианту разработать функции, рисующие следующие геометрические фигуры:

- незакрашенную фигуру (фигуру-контур);
- закрашенную фигуру;
- две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры.

Включить в программу проверки корректности данных (нулевой радиус окружности, нарушение неравенства треугольника и т. д.), в том числе проверки нахождения фигуры в пределах окна и вложенности двух фигур.

Разработать программу так, чтобы в ней был определен класс, реализующий понятие геометрической фигуры в графической системе. Определить ответственность класса. Учесть, что общение с пользователем (включая ввод данных с клавиатуры и вывод данных на экран, если их предполагается использовать), а также реакцию на возникающие при работе с функциями класса ошибки следует производить вне функций класса. Определить атрибуты, необходимые для реализации класса. Поместить атрибуты в закрытую часть описания. Определить функции, необходимые для реализации класса. Выделить интерфейс класса и поместить его в открытую часть описания.

Включить в разработанный класс функции:

- устанавливающие и изменяющие геометрические и графические характеристики фигуры (set-функции);
- возвращающие геометрические и графические характеристики фигуры (get-функции);
- рисующие фигуру на экране;
- изменяющие положение фигуры на экране;
- обеспечивающие сохраняемость объекта: сохранение набора атрибутов объекта класса в файле и считывание его из файла (файлы для сохранения и считывания должны иметь один формат).

Обработку ошибок (нулевой радиус окружности, нарушение неравенства треугольника, другие ошибки задания фигуры, ошибки работы с графикой и др.) осуществлять с использованием механизма возбуждения и обработки исключительных ситуаций.

Разработать функцию, демонстрирующую поведение класса. Поместить реализацию класса в один файл, а демонстрационную функцию – в другой. Обеспечить необходимые межмодульные связи.

Подготовить текстовый файл с разработанной программой, оттранслировать, собрать и выполнить программу с учетом требований операционных систем и программных оболочек, в которых эта программа выполняется. При необходимости исправить ошибки и вновь повторить технологический процесс решения задачи.

Оформить отчет по лабораторной работе. Отчет должен содержать постановку задачи, алгоритм, описание разработанных функций, текст разработанной программы и результаты тестирования.

Защитить лабораторную работу, ответив на вопросы преподавателя.

Вариант задания №2 – выпуклый четырёхугольник.

2. Структурное описание

В данной лабораторной работе мы реализуем класс **’’Quadrangle.h’’**. Для удобства будем называть его «класс».

Для реализации данного класса созданы три файла. Заголовочный файл и два файла исходного кода. В заголовочном файле ’’Quadrangle.h’’ описывается интерфейс (тело) класса, в файле “Quadrangle_methods.cpp” реализованы методы класса.

Класс содержит в себе закрытые (private) члены класса:

- атрибуты, необходимые для реализации класса:
  - структура **POINT** (имеет два целочисленных поля x, y), в которой хранятся координаты текущего значения пера;
  - структуры **top**, в которых хранятся координаты вершин четырёхугольника (x и y);
- функции проверки корректности данных:
  - **isBulge()** - проверка на выпуклость многоугольника;

- **isOnNest()** – проверка на вложенность внутреннего многоугольника во внешний.

В открытые (public) члены класса входят:

- конструктор класса по умолчанию **Quadrangle()**;
- деструктор **~Quadrangle()**;
- конструктор с параметрами **Quadrangle(…)**;
- функция, устанавливающая и изменяющая геометрические характеристики фигуры **setQuadrangle()**;
- функция возвращающая геометрические характеристики фигуры **getQuadrangle()**;
- функция получения данных (координат четырёхугольника) из файла **enterCoordinatesFromFiles()**;
- функция сохранения данных (координат четырёхугольника) в файл **saveCoordinatesToFile ()**;
- функция, сдвигающую фигуру на экране на заданное смещение **changePosition()**.

Общение с пользователем, т. е. рисование фигур должно производиться вне функций основного класса. Для этого создадим отдельный класс **’’View.h’’**.

Для реализации данного класса созданы три файла. Заголовочный файл и два файла исходного кода. В заголовочном файле ’’View.h’’ описывается интерфейс (тело) класса, в файле “View _methods.cpp” реализованы методы класса.

Класс содержит в себе закрытые (private) члены класса:

- атрибуты, необходимые для реализации класса:
  - **HWND hwnd** - идентификатор окна;
  - **HDC hdc** - контекст изображения;
  - **RECT rt** - структура окна с полями left, top, right, buttom;
  - **HPEN hPen** – перо;
  - **HBRUSH hBrush** – кисть;
  - **color colorPen** – структура, в которой хранится цвет пера (в формате RGB);
  - **color colorBrush** - структура, в которой хранится цвет кисти (в формате RGB);
  - **short width** - ширина кисти.
- функции проверки корректности данных:
  - **isAffiliation()** - проверки нахождения фигуры в пределах окна;
  - **isRGB()** – проверка правильности установленного цвета;
  - **isWidth()** - проверка правильности установленной толщины пера.

В открытые (public) члены класса входят:

- конструктор класса по умолчанию **View ()**;
- деструктор **~View ()**;
- конструктор с параметрами **View (…)**;
- функция, устанавливающая и изменяющая графические характеристики фигуры **setCharacteristicsView()**;
- функция, возвращающая графическую характеристику фигуры - ширина кисти;
- функция, возвращающая графическую характеристику фигуры - цвет пера;
- функция, возвращающая графическую характеристику фигуры - цвет кисти;
- функцию, выводящую на консоль не закрашенную фигуру **viewUnpaintQudrangle()**;
- функцию, выводящую на консоль закрашенную фигуру **viewPaintQudrangle()**;
- функцию, выводящую на консоль две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры **viewTwoNestedPaintQuadrangle()**;
- функцию получения данных (графических параметров четырёхугольника) из файла **enterCharacteristicsFromFiles ()**;
- функцию сохранения данных (графических параметров четырёхугольника) в файл **saveCharacteristicsToFile ()**.

В файле “main.cpp” размещена функция int main(), которая демонстрирует поведение класса.

3. Функциональное описание

Рассмотрим реализацию методов класса.

Подключаем заголовочный файл ’’Quadrangle.h’’.

Private:

- **bool isBulge(top leftTop, top rightTop, top rightBottom, top leftBottom)** - проверка на выпуклость многоугольника, выполняется как дружественная функция при выводе на консоль изображения четырёхуголника. Метод принимает в качестве параметров координаты четырёхуголника (4 структуры типа top) и проверяет, чтобы все углы при последовательном обходе четырёхугольника имели один знак. Знак узнаётся через векторное произведение векторов двух соседних сторон;
- **bool isOnNest(Quadrangle* internalQuadrangle)** - проверка на вложенность внутреннего многоугольника во внешний. Проводится путём сравнения координат внутреннего и внешнего четырёхугольников. Метод получает в качестве фактического параметра объект класса – внутренний четырёхугольник. Далее, с помощью this получаются координаты текущего (внешнего) и полученного (внутреннего) четырёхугольников. Путём сравнения их между собой происходит проверка. При нарушении вложенности создаётся сообщение "Внутренний четырёхугольник выходит за границы внешнего!". Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника.

Public:

- конструктор класса по умолчанию **Quadrangle ()** – запускается автоматически при создании нового объекта класса без параметров, создаёт динамический массив структур POINT и инициализирует поля этой структуры некими значениями по умолчанию;
- деструктор **~ Quadrangle ()** – запускается автоматически при удалении объекта класса и освобождает выделенную для создания объекта класса динамическую память;
- конструктор с параметрами **Quadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom)** – данный конструктор принимает в качестве параметров 4 структуры типа top (координаты четырёхугольника) и с помощью оператора new создаёт динамический массив структур POINT, который инициализирует поля этой структуры значениями, полученными в качестве фактических параметров метода;
- **void *setQuadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom) -** принимает в качестве параметров 4 структуры типа top (координаты четырёхугольника). Далее метод инициализирует поля этой структуры значениями, полученными в качестве фактических параметров метода. При этом происходит проверка на выпуклость полученного четырёхугольника. Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;
- **POINT *getQuadrangle()** - метод не принимает никаких параметров, в качестве результата возвращает указатель на динамический массив структур POINT. Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;
- **void enterCoordinatesFromFiles(string fd)** – метод получает данные (координаты четырёхугольника) из файла, указанного в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл открывается, из него считываются координаты, происходит проверка координат на условие выпуклости четырёхугольника, и файл закрывается. Если в файле отсутствуют требуемые координаты, создаётся сообщение " В этом файле нет нужных данных!";
- **void saveCoordinatesToFile (string fd)** – метод сохраняет данные (координаты четырёхугольника) в файл, название которого задаётся в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл создаётся, в него записываются координаты через пробел, и файл закрывается;
- **void changePosition(int x, int y)** – метод сдвигает фигуру на экране на заданное смещение. В качестве фактических параметров метод принимает координаты, на которые нужно переместить фигуру. В теле метода вызывается метод setQuadrangle, которому в качестве фактических параметров передаются координаты, смещённые на указанные позиции.

Подключаем заголовочный файл ’’View.h’’.

Private:

- **bool isAffiliation(Quadrangle* quadrangle)** – метод проверки нахождения фигуры в пределах окна. Для определения размера области окна, в которую можно осуществлять вывод (точнее, ее координат в пикселях), предназначена функция GetClientRect, входящая в стандартную библиотеку <Windows.h>. Вызвав её, определяем фактические размеры окна - int maxX= rt.right, maxY = rt.bottom. В качестве фактического параметра метод принимает объект класса. С помощью метода getQuadrangle() получим значение координат этого объекта класса. Сравнивая координаты объекта и координаты окна, можно выполнить проверку. В случае, если координаты объекта находятся вне области окна создаётся сообщение "Четырёхугольник выходит за границу экрана!";
- **bool isRGB(color colorPenOrBrush)** – метод проверки правильности установленного цвета. В качестве фактических параметров метод принимает значения цветов в формате RGB (структуру типа color). В теле метода происходит проверка принадлежности цветов области значений 0..255. В случае, если указанные значения не входят в данную область создаётся сообщение "Такого цвета не существует!";
- **bool isWidth(short *width)** – метод проверки правильности установленной толщины пера. В качестве фактических параметров метод принимает целочисленное значение пера. В теле метода происходит проверка принадлежности пера области значений 0..9. В случае, если указанное значение не входит в данную область создаётся сообщение "Такую толщину линий задать нельзя! Максимальная толщина - 9 пикселей!".

Public:

- конструктор класса по умолчанию **View()** – запускается автоматически при создании нового объекта класса без параметров, возвращает контекст изображения окна с идентификатором, инициализирует параметры кисти и заливки некими значениями по умолчанию;
- деструктор **~View()** – запускается автоматически при удалении объекта класса и освобождает контекст отображения, полученный для окна. Также выбирает в контекст отображения другие перо и заливку и удаляет текущие перо и заливку;
- конструктор с параметрами **View(HWND, HDC, short *width, color *colorPen, color *colorBrush)** –данный конструктор принимает в качестве параметров окно, контекст отображения, толщину пера, значение цвета пера, значение цвета заливки и инициализирует ими объект. При этом происходит проверка корректности полученных значений – вызываются методы checkRGB() и checkWidth(). Далее метод устанавливает стиль, ширину и цвет пера и цвет заливки;
- **void setCharacteristicsView(short *width, color *colorPen, color *colorBrush)** – метод устанавливает и изменяет графические характеристики фигуры. Принимает в качестве параметров толщину пера, значение цвета пера, значение цвета заливки. Метод устанавливает полученные значения для текущего объекта. При этом происходит проверка корректности полученных значений – вызываются методы isRGB() и isWidth(). Далее метод устанавливает стиль, ширину и цвет пера и цвет заливки;
- **short *getWidth()** - метод не принимает никаких параметров, в качестве результата возвращает указатель на целочисленное значение ширины кисти (переменной width). Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника;
- **color *getColorPen()** - метод не принимает никаких параметров, в качестве результата возвращает указатель на структуру цвета кисти (color colorPen). Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника**;**
- **color *getColorBrush()** - метод не принимает никаких параметров, в качестве результата возвращает указатель на структуру цвета заливки (color colorBrush). Выполняется как дружественная функция при выводе на консоль изображения четырёхугольника**;**
- **void viewUnpaintQudrangle(Quadrangle* quadrangle)** – метод, выводит на консоль не закрашенную фигуру. Сначала происходит проверка нахождения фигуры в пределах окна. Далее указывается прямоугольник для перерисовки окна и обновляется содержимое окна. Потом выбираются кисть и заливка в контекст отображения. Заливка выбирается пустая, типа NULL_BRUSH. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>;
- **void viewPaintQudrangle(Quadrangle* quadrangle)** – метод, выводит на консоль нзакрашенную фигуру. Сначала происходит проверка нахождения фигуры в пределах окна. Далее указывается прямоугольник для перерисовки окна и обновляется содержимое окна. Потом выбираются кисть и за-

ливка в контекст отображения. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>;

- **void viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle *internalQuadrangle)** – метод, выводит на консоль две вложенные одна в другую фигуры, внешняя фигура закрашена, за исключением пространства внутренней фигуры. Сначала происходит проверка нахождения фигур в пределах окна. Далее указывается прямоугольник для перерисовки окна и обновляется содержимое окна. Потом выбираются кисть и заливка в контекст отображения. При этом для обоих четырёхугольников заливка выбирается пустая, типа NULL_BRUSH. Затем получаются координаты объекта, получаемого в качестве фактического параметра с помощью дружественной функции getQuadrangle(). И, наконец, происходит вывод на консоль изображения четырёхугольника с помощью стандартной функции Polygon() из стандартной библиотеки <Windows.h>. Далее происходит закраска внутренней области внешнего четырёхугольника с помощью стандартной функции FloodFill(), для которой указывается точка, находящаяся внутри замкнутой области;
- **void enterCharacteristicsFromFiles(string fd)** – метод получает данные (графических параметров четырёхугольника) из файла, указанного в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл открывается, из него считываются координаты, происходит проверка координат на условия корректности данных, и файл закрывается. Если в файле отсутствуют требуемые координаты, создаётся сообщение " В этом файле нет нужных данных!";
- **void saveCharacteristicsToFile(string fd)** – метод сохраняет данные (графических параметров четырёхугольника) в файл, название которого задаётся в качестве фактического параметра. Для этого с помощью функций из стандартной библиотеки <fstream> файл создаётся, в него записываются координаты через пробел, и файл закрывается.

4. Описание работы программы

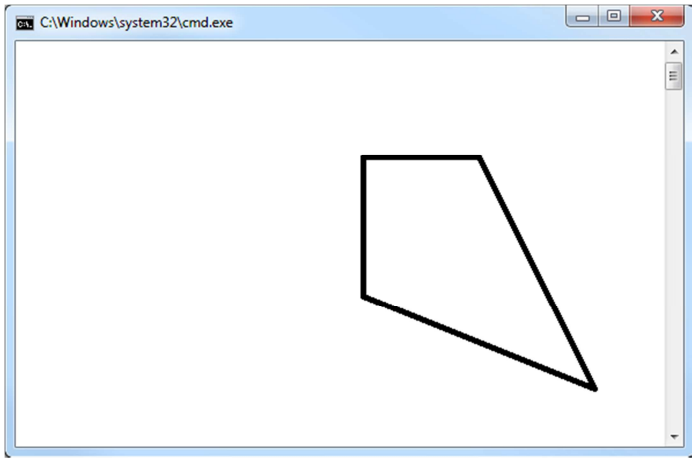

Рис. 1 – Вывод в консоль не закрашенной фигуры (фигуры-контура).

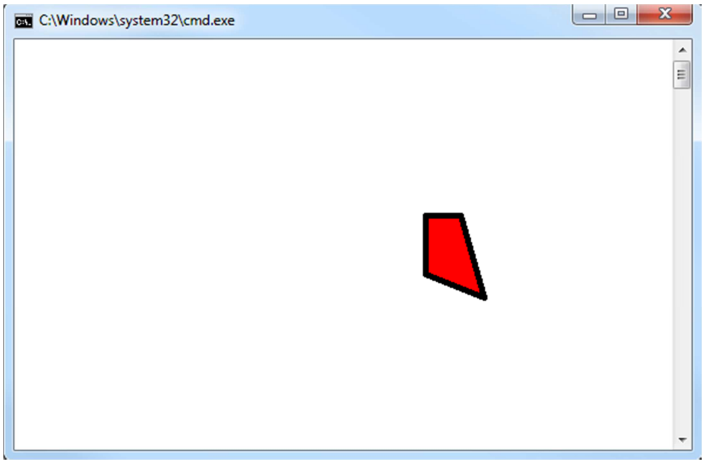

Рис. 2 – Вывод в консоль закрашенной фигуры.

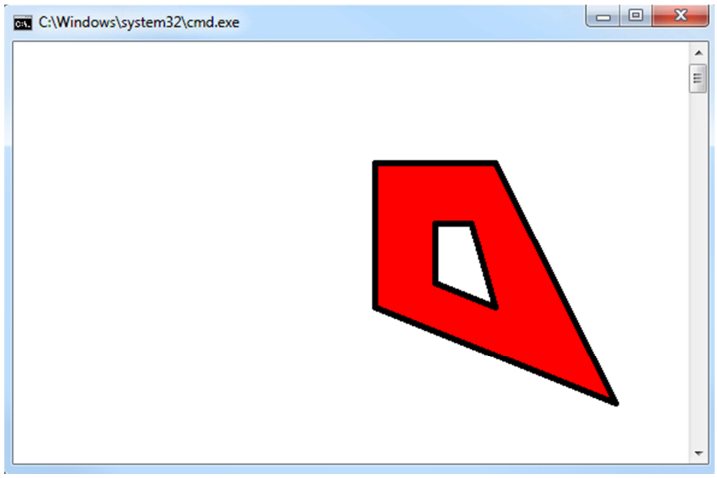

Рис. 3 - Вывод в консоль двух вложенных одна в другую фигуры (внешняя фигура закрашена за исключением пространства внутренней фигуры).

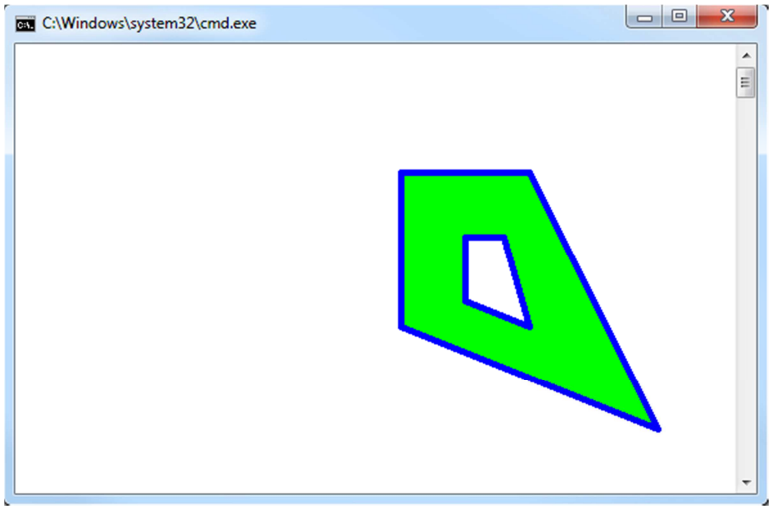

Рис. 4 - Вывод в консоль двух вложенных одна в другую фигуры (внешняя фигура закрашена за исключением пространства внутренней фигуры) с новыми графическими параметрами, полученными из файла.

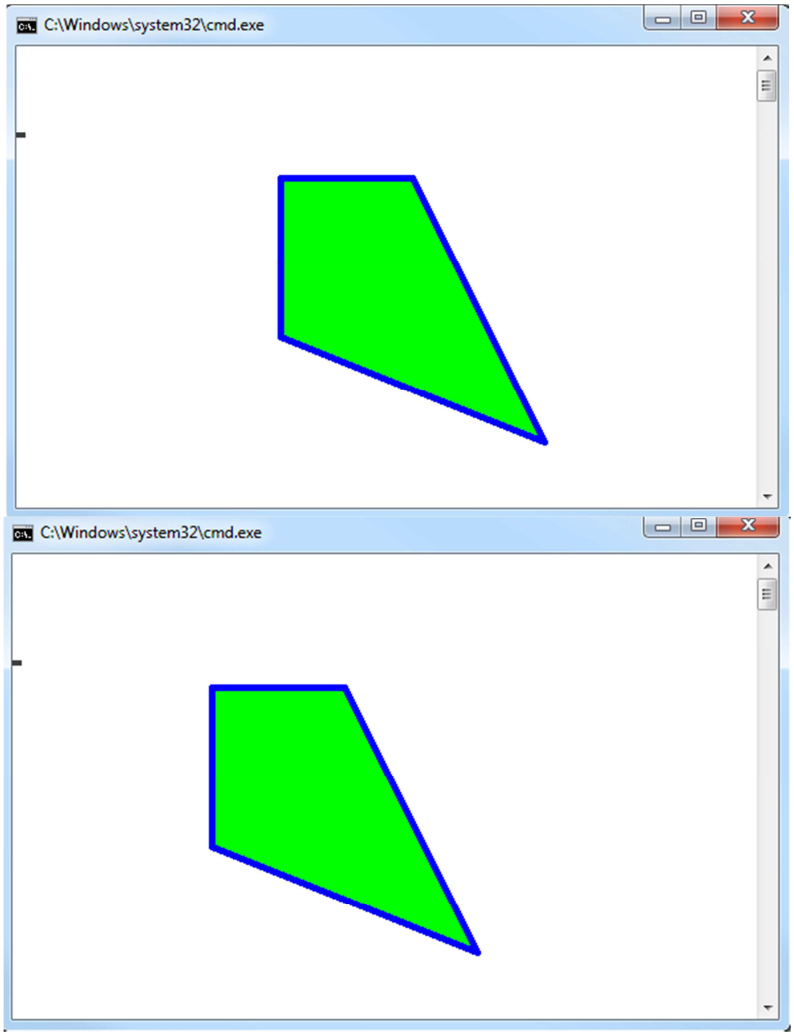

Рис. 5 - Демонстрация работы функции сдвига фигуры.

Выводы

В ходе выполнения лабораторной работы были изучены основные понятия и функции графического интерфейса операционной системы Microsoft Windows.

Также были изучены следующие понятия: определение типа данных пользователя с помощью конструкции class, открытая и закрытая части описания класса, определение набора функций класса, интерфейс и реализация класса, реализация понятия модульности в языке C++.

Ссылка на github.com: **https://github.com/IvanSokotushenko/OOP/tree/master/Lab1var2v11**.

Приложение. Текст программы

```
#pragma once
#include <windows.h>
#include <iostream>
#include <fstream>
using namespace std;

struct top
{
	int x, y;
};

class Quadrangle
{
	POINT *points;
	top leftTop, rightTop, rightBottom, leftBottom, change;
	bool isBulge(top leftTop, top rightTop, top rightBottom, top leftBottom);
	bool isOnTheNested(Quadrangle *internalQuadrangle);
	friend class View;
public:
	Quadrangle();
	~Quadrangle();
	Quadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom);
	POINT *setQuadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom);
	POINT *getQuadrangle();
	void enterCoordinatesFromFiles(string fd);
	void saveCoordinatesToFile(string fd);
	void changePosition(top *change);
};

#include "Quadrangle.h"

bool Quadrangle::isBulge(top leftTop, top rightTop, top rightBottom, top leftBottom)
{
return (((rightTop.x - leftTop.x)*(rightBottom.y - leftTop.y) - (rightTop.y - leftTop.y)*
	(rightBottom.x - leftTop.x))*((rightTop.x - leftTop.x)*(leftBottom.y - leftTop.y) –
	(rightTop.y - leftTop.y)*(leftBottom.x - leftTop.x))<0 ||
	((leftBottom.x - rightBottom.x)*(rightTop.y - rightBottom.y) - (leftBottom.y - rightBot-
	tom.y)*(rightTop.x - rightBottom.x))*((leftBottom.x - rightBottom.x)*
	(leftTop.y - rightBottom.y) - (leftBottom.y - rightBottom.y)*(leftTop.x - rightBottom.x))<0);
}

bool Quadrangle::isOnTheNested(Quadrangle *internalQuadrangle)
{
	POINT *points1 = this->getQuadrangle();
	POINT *points2 = internalQuadrangle->getQuadrangle();
	return ((points2[0].x < points1[0].x) || (points2[0].y < points1[0].y) ||
		(points2[1].x > points1[1].x) || (points2[1].y < points1[1].y) ||
		(points2[2].x > points1[2].x) || (points2[2].y > points1[2].y) ||
		(points2[3].x < points1[3].x) || (points2[3].y > points1[3].y));
}

POINT *Quadrangle::setQuadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom)
{
	if (isBulge(leftTop, rightTop, rightBottom, leftBottom))
		throw exception("Четырёхугольник не является выпуклым!");
	points[0] = { leftTop.x, leftTop.y };
	points[1] = { rightTop.x, rightTop.y };
	points[2] = { rightBottom.x, rightBottom.y };
	points[3] = { leftBottom.x, leftBottom.y };
}

POINT* Quadrangle::getQuadrangle()
{
	return this->points;
}
```

```
Quadrangle::Quadrangle()
{
	points = new POINT[4];
	points[0] = { 300, 100 };
	points[1] = { 400, 100 };
	points[2] = { 300, 200 };
	points[3] = { 500, 300 };
}

Quadrangle::~Quadrangle()
{
	delete[] getQuadrangle();
}

Quadrangle::Quadrangle(top leftTop, top rightTop, top rightBottom, top leftBottom)
{
	if (isBulge(leftTop, rightTop, rightBottom, leftBottom))
		throw exception("Четырёхугольник не является выпуклым!");
	points = new POINT[4];
	points[0] = { leftTop.x, leftTop.y };
	points[1] = { rightTop.x, rightTop.y };
	points[2] = { rightBottom.x, rightBottom.y };
	points[3] = { leftBottom.x, leftBottom.y };
}

void Quadrangle::enterCoordinatesFromFiles(string fd)
{
	ifstream in(fd, ios::in);
	if (in.is_open())
	{
		POINT *points = this->getQuadrangle();
		in >> points[0].x;
		in >> points[0].y;
		in >> points[1].x;
		in >> points[1].y;
		in >> points[2].x;
		in >> points[2].y;
		in >> points[3].x;
		in >> points[3].y;
		if (isBulge(leftTop, rightTop, rightBottom, leftBottom))
			throw exception("Четырёхугольник не является выпуклым!");
		in.close();
	}
	else
		throw exception("В этом файле нет нужных данных!");
}

void Quadrangle::saveCoordinatesToFile(string fd)
{
	ofstream out(fd, ios::out);
	out << this->points[0].x;
	out << " ";
	out << this->points[0].y;
	out << " ";
	out << this->points[1].x;
	out << " ";
	out << this->points[1].y;
	out << " ";
	out << this->points[2].x;
	out << " ";
	out << this->points[2].y;
	out << " ";
	out << this->points[3].x;
	out << " ";
	out << this->points[3].y;
	out << " ";
	out.close();
}
```

```
void Quadrangle::changePosition(top *change)
{
	this->change = *change;
	setQuadrangle({ this->points[0].x + this->change.x, points[0].y + this->change.y },
			{ points[1].x + this->change.x, points[1].y + this->change.y },
			{ points[2].x + this->change.x, points[2].y + this->change.y },
			{ points[3].x + this->change.x, points[3].y + this->change.y });
}
#pragma once
#include <windowsx.h>
#include "Quadrangle.h"

struct color
{
	short R;
	short G;
	short B;
};

class View
{
	HWND hwnd;
	HDC hdc;
	RECT rt;
	HPEN hPen;
	HBRUSH hBrush;
	color colorPen, colorBrush;
	short width;

	bool isAffiliation(Quadrangle *quadrangle);
	bool isRGB(color colorPenOrBrush);
	bool isWidth(short *width);

public:
	View();
	~View();
	View(HWND, HDC, short *width, color *colorPen, color *colorBrush);
	void setCharacteristicsView(short *width, color *colorPen, color *colorBrush);
	short *getWidth();
	color *getColorPen();
	color *getColorBrush();
	void viewUnpaintQudrangle(Quadrangle *quadrangle);
	void viewPaintQudrangle(Quadrangle *quadrangle);
	void viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle
*internalQuadrangle);
	void enterCharacteristicsFromFiles(string fd);
	void saveCharacteristicsToFile(string fd);
};

#include "View.h"

bool View::isAffiliation(Quadrangle *quadrangle)
{
	GetClientRect(this->hwnd, &(this->rt));
	POINT *points = quadrangle->getQuadrangle();
	return ((points[0].x < rt.left) || (points[0].y < rt.top) || (points[0].x > rt.right) ||
		(points[0].y > rt.bottom) || (points[1].x > rt.right) || (points[1].y < rt.top) ||
		(points[1].x < rt.left) || (points[1].y > rt.bottom) ||	(points[2].x > rt.right)
		|| (points[2].y > rt.bottom) || (points[2].x < rt.left) || (points[2].y < rt.top) ||
		(points[3].x < rt.left) || (points[3].y > rt.bottom) || (points[3].x > rt.right) ||
		(points[3].y < rt.top));
}

bool View::isRGB(color colorPenOrBrush)
{
	return ((colorPenOrBrush.R < 0 || colorPenOrBrush.R > 255) ||
			(colorPenOrBrush.G < 0 || colorPenOrBrush.G > 255) ||
			(colorPenOrBrush.B < 0 || colorPenOrBrush.B> 255));
}
```

```
bool View::isWidth(short *width)
{
	return (*width < 0 || *width > 9);
}

View::View()
{
	hwnd = GetConsoleWindow();
	hdc = GetDC(hwnd);
	colorPen = { 0, 0, 0 };
	colorBrush = { 255, 255, 255 };
	hPen = CreatePen(PS_SOLID, 5, RGB(this->colorPen.R, this->colorPen.G, this->colorPen.B));
	hBrush = CreateSolidBrush(RGB(this->colorBrush.R, this->colorBrush.G, this->colorBrush.B));
}

View::~View()
{
	SelectPen(hdc, hPen);
	SelectBrush(hdc, hBrush);
	DeletePen(hPen);
	DeleteBrush(hBrush);
	ReleaseDC(this->hwnd, this->hdc);
}

View::View(HWND hwnd, HDC hdc, short *width, color *colorPen, color *colorBrush)
{
	this->hwnd = hwnd;
	this->hdc = hdc;
	this->width = *width;
	if (isWidth(width))
		throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пиксе-
лей.");
	this->colorPen = *colorPen;
	if (isRGB(*colorPen))
		throw exception("Такого цвета не существует!");
	this->colorBrush = *colorBrush;
	if (isRGB(*colorBrush))
		throw exception("Такого цвета не существует!");
	hPen = CreatePen(PS_SOLID, this->width, RGB(this->colorPen.R, this->colorPen.G, this-
>colorPen.B));
	hBrush = CreateSolidBrush(RGB(this->colorBrush.R, this->colorBrush.G, this->colorBrush.B));
}

void View::setCharacteristicsView(short *width, color *colorPen, color *colorBrush)
{
	this->width = *width;
	if (isWidth(width))
		throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9 пиксе-
лей.");
	this->colorPen = *colorPen;
	if (isRGB(*colorPen))
		throw exception("Такого цвета не существует!");
	this->colorBrush = *colorBrush;
	if (isRGB(*colorBrush))
		throw exception("Такого цвета не существует!");
	hPen = CreatePen(PS_SOLID, this->width, RGB(this->colorPen.R, this->colorPen.G, this-
>colorPen.B));
	hBrush = CreateSolidBrush(RGB(this->colorBrush.R, this->colorBrush.G, this->colorBrush.B));
}

short *View::getWidth()
{
	return &this->width;
}
color *View::getColorPen()
{
	return &this->colorPen;
}
```

```
color *View::getColorBrush()
{
	return &this->colorBrush;
}

void View::viewUnpaintQudrangle(Quadrangle *quadrangle)
{
	if (isAffiliation(quadrangle))
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, GetStockBrush(NULL_BRUSH));
	POINT *points = quadrangle->getQuadrangle();
	Polygon(this->hdc, points, 4);
}

void View::viewPaintQudrangle(Quadrangle *quadrangle)
{
	if (isAffiliation(quadrangle))
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, this->hBrush);
	POINT *points = quadrangle->getQuadrangle();
	Polygon(this->hdc, points, 4);
}

void View::viewTwoNestedPaintQuadrangle(Quadrangle *externalQuadrangle, Quadrangle
*internalQuadrangle)
{
	if (isAffiliation(externalQuadrangle))
		throw exception("Четырёхугольник выходит за границу экрана!");
	InvalidateRect(this->hwnd, 0, TRUE);
	UpdateWindow(this->hwnd);
	if (externalQuadrangle->isOnTheNested(internalQuadrangle))
		throw exception("Внутренний четырёхугольник выходит за границы внешнего!");
	HPEN SelectPen(this->hdc, this->hPen);
	HBRUSH SelectBrush(this->hdc, GetStockBrush(NULL_BRUSH));
	POINT *points1 = externalQuadrangle->getQuadrangle();
	POINT *points2 = internalQuadrangle->getQuadrangle();
	Polygon(this->hdc, points1, 4);
	Polygon(this->hdc, points2, 4);
	HBRUSH SelectBrush(this->hdc, this->hBrush);
	FloodFill(this->hdc, points1[0].x + 10, points1[0].y + 10, RGB(this->colorPen.R, this-
>colorPen.G, this->colorPen.B));
}
```

```
void View::enterCharacteristicsFromFiles(string fd)
{
	ifstream in(fd, ios::in);
	if (in.is_open())
	{
		in >> this->width;
		if (isWidth(&this->width))
			throw exception("Такую толщину линий задать нельзя!\nМаксимальная толщина - 9
пикселей.");
		in >> this->colorPen.R;
		in >> this->colorPen.G;
		in >> this->colorPen.B;
		this->colorPen = colorPen;
		if (isRGB(colorPen))
			throw exception("Такого цвета не существует!");
		in >> this->colorBrush.R;
		in >> this->colorBrush.G;
		in >> this->colorBrush.B;
		if (isRGB(colorBrush))
			throw exception("Такого цвета не существует!");
		this->hPen = CreatePen(PS_SOLID, 5, RGB(this->colorPen.R, this->colorPen.G, this-
>colorPen.B));
		this->hBrush = CreateSolidBrush(RGB(this->colorBrush.R, this->colorBrush.G, this-
>colorBrush.B));
		in.close();
	}
	else
		throw exception("В этом файле нет нужных данных");
}

void View::saveCharacteristicsToFile(string fd)
{
	ofstream out(fd, ios::out);
	out << this->width;
	out << " ";
	out << this->colorPen.R;
	out << " ";
	out << this->colorPen.G;
	out << " ";
	out << this->colorPen.B;
	out << " ";
	out << this->colorBrush.R;
	out << " ";
	out << this->colorBrush.G;
	out << " ";
	out << this->colorBrush.B;
	out.close();
}
#include "Quadrangle.h"
#include "View.h"
```

```
int main()
{
	setlocale(LC_ALL, "rus");
	try {
		HWND hwnd = GetConsoleWindow();
		HDC hdc = GetDC(hwnd);

		color colorPen = { 0, 0, 0 };
		color colorBrush = { 255, 0, 0 };
		short width = 5;

		// Создаём три четырёхугольника
		Quadrangle first({ 300, 100 }, { 400, 100 }, { 500, 300 }, { 300, 220 });
		Quadrangle second({ 350, 150 }, { 380, 150 }, { 400, 220 }, { 350, 200 });
		Quadrangle third({ 0, 0 }, { 0, 0 }, { 0, 0 }, { 0, 0 });
		third.enterCoordinatesFromFiles("Quadrangle_CoordinatesFromFile.txt");
		// Вызываем конструктор с параметрами
		View quadrangle1(hwnd, hdc, &width, &colorPen, &colorBrush);
		// Выводим изображения
		quadrangle1.viewUnpaintQudrangle(&first);
		getchar();
		quadrangle1.viewPaintQudrangle(&second);
		getchar();
		quadrangle1.viewTwoNestedPaintQuadrangle(&first, &second);
		getchar();
		// Выводим изображение двух вложенных четырёхугольников с новыми параметрами, получен-
		ными из файла
		quadrangle1.enterCharacteristicsFromFiles("Quadrangle_CharacteristicsFromFiles.txt");
		quadrangle1.viewTwoNestedPaintQuadrangle(&first, &second);
		getchar();
		// Демонстрация работы функции сдвига фигуры
		quadrangle1.viewPaintQudrangle(&third);
		getchar();
		top change = { -100, 0 };
		third.changePosition(&change);
		quadrangle1.viewPaintQudrangle(&third);
		getchar();
		third.saveCoordinatesToFile("Quadrangle_CoordinatesInFile.txt");
		quadrangle1.saveCharacteristicsToFile("Quadrangle_CharacteristicsInFiles.txt");
	}
	catch (exception exeption)
	{
		cout << exeption.what() << endl;
	}
	return 0;
}
```