МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

По объектно-ориентированному программированию

«Разработка графического редактора на языке C++ с применением механизмов ООП»

Выполнила:

студентка гр. ПМ-21-2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бутусова В.М.

Проверил:

доц., к.п.н. кафедры АСУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кургасов В. В.

Липецк 2022

Оглавление

[Цель работы. 2](#_Toc122472872)

[Задание кафедры. 3](#_Toc122472873)

[Вариант №3. 4](#_Toc122472874)

[Код программы. 5](#_Toc122472875)

[Пример работы программы. 14](#_Toc122472876)

[Вывод. 20](#_Toc122472877)

[Контрольные вопросы. 21](#_Toc122472878)

Цель работы.

Закрепить навыки использования механизмов ООП на примере реализации графического редактора.

Задание кафедры.

Реализовать на языке C++ редактор графической схемы. В ходе выполненной работы обязательно применение объектно-ориентированных возможностей языка C++: наследования и динамического полиморфизма. Каждый тип элемента схемы должен быть представлен в программе в виде отдельного класса, который наследован от базового класса «графический элемент» (имеющего чисто виртуальную функцию прорисовки). Также необходим один класс «поле рисования», который содержит все графические элементы и отвечает за вызов функций прорисовки. Хранение графических элементов осуществляется с использованием контейнеров стандартной библиотеки C++.

Вариант №3.

|  |  |
| --- | --- |
| Вариант | Редактор |
| 3 | Векторный графический редактор. |

Код программы.

namespace Redact {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Collections::Generic;

public ref class Redactor : public System::Windows::Forms::Form // основной класс, через который будет запускаться вся программа

{

public:

Redactor(void)

{

InitializeComponent(); //инициалиация компонентов

openFileDialog1->Filter = "Файлы изображения (\*.bmp, \*.jpg, \*.png)|\*.bmp; \*.jpg; \*.png";

saveFileDialog1->Filter = "Файлы изображения (\*.bmp, \*.jpg, \*.png)|\*.bmp; \*.jpg; \*.png";

}

protected:

~Redactor()

{

if (components)

{

delete components;

}

}

protected:

private: System::Windows::Forms::PictureBox^ pictureBox1;

private: System::Windows::Forms::GroupBox^ groupBox1;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::NumericUpDown^ numericUpDown1;

private: System::Windows::Forms::Button^ Ластик;

private: System::Windows::Forms::Button^ Кисть;

private: System::Windows::Forms::GroupBox^ groupBox2;

private: System::Windows::Forms::Button^ линия;

private: System::Windows::Forms::Button^ цвет2;

private: System::Windows::Forms::Button^ круг;

private: System::Windows::Forms::Button^ квадрат;

private: System::Windows::Forms::ColorDialog^ colorDialog1;

private: System::Windows::Forms::OpenFileDialog^ openFileDialog1;

private: System::Windows::Forms::SaveFileDialog^ saveFileDialog1;

private: System::Windows::Forms::Button^ clear\_all;

private: System::Windows::Forms::Button^ OK;

private: System::Windows::Forms::Button^ save\_picture;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(Redactor::typeid));

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());

this->OK = (gcnew System::Windows::Forms::Button());

this->label1 = (gcnew System::Windows::Forms::Label());

this->numericUpDown1 = (gcnew System::Windows::Forms::NumericUpDown());

this->Ластик = (gcnew System::Windows::Forms::Button());

this->Кисть = (gcnew System::Windows::Forms::Button());

this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());

this->линия = (gcnew System::Windows::Forms::Button());

this->цвет2 = (gcnew System::Windows::Forms::Button());

this->круг = (gcnew System::Windows::Forms::Button());

this->квадрат = (gcnew System::Windows::Forms::Button());

this->colorDialog1 = (gcnew System::Windows::Forms::ColorDialog());

this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());

this->saveFileDialog1 = (gcnew System::Windows::Forms::SaveFileDialog());

this->clear\_all = (gcnew System::Windows::Forms::Button());

this->save\_picture = (gcnew System::Windows::Forms::Button());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

this->groupBox1->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->BeginInit();

this->groupBox2->SuspendLayout();

this->SuspendLayout();

//

// pictureBox1

//

this->pictureBox1->BackColor = System::Drawing::Color::White;

this->pictureBox1->BorderStyle = System::Windows::Forms::BorderStyle::FixedSingle;

this->pictureBox1->Location = System::Drawing::Point(322, 12);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(941, 577);

this->pictureBox1->TabIndex = 2;

this->pictureBox1->TabStop = false;

this->pictureBox1->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &Redactor::pictureBox1\_Paint);

this->pictureBox1->MouseDown += gcnew System::Windows::Forms::MouseEventHandler(this, &Redactor::pictureBox1\_MouseDown);

this->pictureBox1->MouseMove += gcnew System::Windows::Forms::MouseEventHandler(this, &Redactor::pictureBox1\_MouseMove);

this->pictureBox1->MouseUp += gcnew System::Windows::Forms::MouseEventHandler(this, &Redactor::pictureBox1\_MouseUp);

//

// groupBox1

//

this->groupBox1->Controls->Add(this->OK);

this->groupBox1->Controls->Add(this->label1);

this->groupBox1->Controls->Add(this->numericUpDown1);

this->groupBox1->Controls->Add(this->Ластик);

this->groupBox1->Controls->Add(this->Кисть);

this->groupBox1->Location = System::Drawing::Point(12, 12);

this->groupBox1->Name = L"groupBox1";

this->groupBox1->Size = System::Drawing::Size(290, 187);

this->groupBox1->TabIndex = 3;

this->groupBox1->TabStop = false;

this->groupBox1->Text = L"Кисть";

//

// OK

//

this->OK->Location = System::Drawing::Point(129, 121);

this->OK->Name = L"OK";

this->OK->Size = System::Drawing::Size(52, 32);

this->OK->TabIndex = 5;

this->OK->Text = L"OK";

this->OK->UseVisualStyleBackColor = true;

this->OK->Click += gcnew System::EventHandler(this, &Redactor::OK\_Click);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(39, 105);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(65, 20);

this->label1->TabIndex = 3;

this->label1->Text = L"Размер";

//

// numericUpDown1

//

this->numericUpDown1->Location = System::Drawing::Point(29, 128);

this->numericUpDown1->Name = L"numericUpDown1";

this->numericUpDown1->Size = System::Drawing::Size(86, 26);

this->numericUpDown1->TabIndex = 2;

this->numericUpDown1->Value = System::Decimal(gcnew cli::array< System::Int32 >(4) { 1, 0, 0, 0 });

//

// Ластик

//

this->Ластик->Location = System::Drawing::Point(120, 36);

this->Ластик->Name = L"Ластик";

this->Ластик->Size = System::Drawing::Size(86, 47);

this->Ластик->TabIndex = 1;

this->Ластик->Text = L"Ластик";

this->Ластик->UseVisualStyleBackColor = true;

this->Ластик->Click += gcnew System::EventHandler(this, &Redactor::Eraser\_button);

//

// Кисть

//

this->Кисть->Location = System::Drawing::Point(15, 36);

this->Кисть->Name = L"Кисть";

this->Кисть->Size = System::Drawing::Size(75, 47);

this->Кисть->TabIndex = 0;

this->Кисть->Text = L"Кисть";

this->Кисть->UseVisualStyleBackColor = true;

this->Кисть->Click += gcnew System::EventHandler(this, &Redactor::Brush\_button);

//

// groupBox2

//

this->groupBox2->Controls->Add(this->линия);

this->groupBox2->Controls->Add(this->цвет2);

this->groupBox2->Controls->Add(this->круг);

this->groupBox2->Controls->Add(this->квадрат);

this->groupBox2->Location = System::Drawing::Point(12, 229);

this->groupBox2->Name = L"groupBox2";

this->groupBox2->Size = System::Drawing::Size(230, 184);

this->groupBox2->TabIndex = 5;

this->groupBox2->TabStop = false;

this->groupBox2->Text = L"Фигуры";

//

// линия

//

this->линия->Location = System::Drawing::Point(15, 103);

this->линия->Name = L"линия";

this->линия->Size = System::Drawing::Size(86, 47);

this->линия->TabIndex = 5;

this->линия->Text = L"Линия";

this->линия->UseVisualStyleBackColor = true;

this->линия->Click += gcnew System::EventHandler(this, &Redactor::line\_button);

//

// цвет2

//

this->цвет2->Location = System::Drawing::Point(120, 110);

this->цвет2->Name = L"цвет2";

this->цвет2->Size = System::Drawing::Size(75, 33);

this->цвет2->TabIndex = 4;

this->цвет2->Text = L"Цвет";

this->цвет2->UseVisualStyleBackColor = true;

this->цвет2->Click += gcnew System::EventHandler(this, &Redactor::color\_line\_button);

//

// круг

//

this->круг->Location = System::Drawing::Point(120, 36);

this->круг->Name = L"круг";

this->круг->Size = System::Drawing::Size(86, 47);

this->круг->TabIndex = 1;

this->круг->Text = L"Круг";

this->круг->UseVisualStyleBackColor = true;

this->круг->Click += gcnew System::EventHandler(this, &Redactor::circle\_button);

//

// квадрат

//

this->квадрат->Location = System::Drawing::Point(15, 36);

this->квадрат->Name = L"квадрат";

this->квадрат->Size = System::Drawing::Size(86, 47);

this->квадрат->TabIndex = 0;

this->квадрат->Text = L"Квадрат";

this->квадрат->UseVisualStyleBackColor = true;

this->квадрат->Click += gcnew System::EventHandler(this, &Redactor::Square\_button);

//

// openFileDialog1

//

this->openFileDialog1->FileName = L"openFileDialog1";

//

// clear\_all

//

this->clear\_all->Location = System::Drawing::Point(58, 438);

this->clear\_all->Name = L"clear\_all";

this->clear\_all->Size = System::Drawing::Size(148, 57);

this->clear\_all->TabIndex = 6;

this->clear\_all->Text = L"Стереть всё";

this->clear\_all->UseVisualStyleBackColor = true;

this->clear\_all->Click += gcnew System::EventHandler(this, &Redactor::clear\_all\_Click);

//

// save\_picture

//

this->save\_picture->Location = System::Drawing::Point(58, 514);

this->save\_picture->Name = L"save\_picture";

this->save\_picture->Size = System::Drawing::Size(148, 57);

this->save\_picture->TabIndex = 7;

this->save\_picture->Text = L"Сохранить рисунок";

this->save\_picture->UseVisualStyleBackColor = true;

this->save\_picture->Click += gcnew System::EventHandler(this, &Redactor::save\_picture\_Click);

//

// Redactor

//

this->AutoScaleDimensions = System::Drawing::SizeF(9, 20);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1278, 601);

this->Controls->Add(this->save\_picture);

this->Controls->Add(this->clear\_all);

this->Controls->Add(this->groupBox2);

this->Controls->Add(this->groupBox1);

this->Controls->Add(this->pictureBox1);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;

//this->Icon = (cli::safe\_cast<System::Drawing::Icon^>(resources->GetObject(L"$this.Icon")));

this->MaximizeBox = false;

this->MaximumSize = System::Drawing::Size(1471, 808);

this->Name = L"Redactor";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"Векторный графический редактор";

this->Load += gcnew System::EventHandler(this, &Redactor::MyForm\_Load);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

this->groupBox1->ResumeLayout(false);

this->groupBox1->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numericUpDown1))->EndInit();

this->groupBox2->ResumeLayout(false);

this->ResumeLayout(false);

}

#pragma endregion

private:

Graphics^ canva; //полотно

List<Point>^ points; // точки для рисования

Pen^ pen; // кисть

bool draw;

Bitmap^ background; // фон

Color color; // цвет

float size; // размер

String^ filename; // имя файла

int statePen; // тип кисти

/\*

1 - линия

2 - квадрат

3 - круг

4 - ластик

\*/

private:

System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) // обработчкк запуска приложения

{

color = Color::Black; // цвет кисти чёртный

size = 1; // размер кисти 1

pen = gcnew Pen(color, size); // создаём кисть и задаём параметры

points = gcnew List <Point>(); // инициализируем точки

statePen = 1; // при запуске рисует кистью

// окрашиваем кнопки

Кисть->BackColor = Color::LightBlue;

Ластик->BackColor = Color::LightGray;

линия->BackColor = Color::LightGray;

квадрат->BackColor = Color::LightGray;

круг->BackColor = Color::LightGray;

//создаём изображение

Bitmap^ Img = gcnew Bitmap(pictureBox1->Width, pictureBox1->Height);

canva = Graphics::FromImage(Img); // запоминаем холст

pictureBox1->Image = Img; // запоминаем озображение

pictureBox1->BackColor = Color::White;

}

System::Void Brush\_button(System::Object^ sender, System::EventArgs^ e) // активация кисти

{

if (statePen != 1) // если не была кисть

{

statePen = 1;

}

color = Color::Black; // цвет - чёрный

pen = gcnew Pen(color, size); // переобределяем кисть

//окрашиваем кнопки

Кисть->BackColor = Color::LightBlue;

Ластик->BackColor = Color::LightGray;

линия->BackColor = Color::LightGray;

квадрат->BackColor = Color::LightGray;

круг->BackColor = Color::LightGray;

}

System::Void Eraser\_button(System::Object^ sender, System::EventArgs^ e) // аткивация ластика

{

if (statePen != 4)

{

{

statePen = 4;

}

}

color = Color::White; // цвет - белый

pen = gcnew Pen(color, size);

//окрашиваем кнопки

Ластик->BackColor = Color::LightBlue;

квадрат->BackColor = Color::LightGray;

линия->BackColor = Color::LightGray;

Кисть->BackColor = Color::LightGray;

круг->BackColor = Color::LightGray;

}

System::Void Square\_button(System::Object^ sender, System::EventArgs^ e) // активация квадрата

{

if (statePen != 2)

{

statePen = 2;

}

квадрат->BackColor = Color::LightBlue;

Ластик->BackColor = Color::LightGray;

линия->BackColor = Color::LightGray;

Кисть->BackColor = Color::LightGray;

круг->BackColor = Color::LightGray;

}

System::Void circle\_button(System::Object^ sender, System::EventArgs^ e) // активация круга

{

if (statePen != 3)

{

statePen = 3;

}

круг->BackColor = Color::LightBlue;

Ластик->BackColor = Color::LightGray;

линия->BackColor = Color::LightGray;

квадрат->BackColor = Color::LightGray;

Кисть->BackColor = Color::LightGray;

}

System::Void line\_button(System::Object^ sender, System::EventArgs^ e) // активация линии

{

if (statePen != 5)

{

statePen = 5;

}

линия->BackColor = Color::LightBlue;

Ластик->BackColor = Color::LightGray;

Кисть->BackColor = Color::LightGray;

квадрат->BackColor = Color::LightGray;

круг->BackColor = Color::LightGray;

}

System::Void color\_line\_button(System::Object^ sender, System::EventArgs^ e) // обработчик выбора цвета

{

colorDialog1->ShowDialog(); // открываем окно цвета

color = colorDialog1->Color; // запоминаем цвет

цвет2->BackColor = color; // окрашиываем кнопку в цвет

pen = gcnew Pen(color, size); // задаём новые параметры кисти

}

System::Void pictureBox1\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) // событие рисования

{

switch (statePen)

{

case 1:

if (points->Count > 1) //рисуем кистью

canva->DrawLines(pen, points->ToArray());

break;

case 2: // рисуем квадрат

canva->DrawRectangle(pen, MousePosition.X - 435, MousePosition.Y - 170, Convert::ToInt32(numericUpDown1->Value), Convert::ToInt32(numericUpDown1->Value));

break;

case 3: // рисуем круг

canva->DrawEllipse(pen, MousePosition.X - 435, MousePosition.Y - 170, Convert::ToInt32(numericUpDown1->Value), Convert::ToInt32(numericUpDown1->Value));

break;

case 4: // рисуем ластиком

if (points->Count > 1)

canva->DrawLines(pen, points->ToArray());

break;

case 5: // рисуем линией

if (points->Count > 1)

canva->DrawLines(pen, points->ToArray());

default:

break;

}

}

System::Void pictureBox1\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) // обработчки события движения мыши

{

if ((draw && points->Contains(Point(e->X, e->Y)) == false) && statePen != 5) // проверка можно ли рисовать, пуст ли список точек, не линия

{

if (e->Button == System::Windows::Forms::MouseButtons::Left) // проверка нажатия левой кнопки мыши

{

points->Add(e->Location); // добавление точки

}

pictureBox1->Invalidate(); // перерисовываем

}

}

System::Void pictureBox1\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) // обработчки события отпускания кнопки мыши

{

draw = false; // рисовать больше нельзя

pictureBox1->Invalidate(); // перерисываем

}

System::Void pictureBox1\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) // обработчк нажатия кнопки мыши

{

if (statePen == 1 || statePen == 4)//если рисуем кистью или ластиком

{

points->Clear(); // очищаем список точек

points->Add(e->Location); // запоминаем все точки

draw = true; // рисовать можно

}

else if (statePen == 5) // если рисуем линию

{

points->Add(e->Location);

draw = true;

}

else

{ // запоминаем позицию мыши

MousePosition.X = e->Location.X;

MousePosition.Y = e->Location.Y;

}

pictureBox1->Invalidate(); // перерисовываем

}

System::Void clear\_all\_Click(System::Object^ sender, System::EventArgs^ e) // обработчки нажатия кнопки очищения холста

{

points->Clear(); // очищаем точки

canva->Clear(Color::White); // очищаем холст

pictureBox1->Invalidate(); // перерисовываем

}

System::Void OK\_Click(System::Object^ sender, System::EventArgs^ e) // обработчик нажатия кнопки OK - новый размер кисти

{

size = Convert::ToInt32(numericUpDown1->Value); // присваиваем новый размер кисти

pen = gcnew Pen(color, size); // переобределяем кисть

}

System::Void save\_picture\_Click(System::Object^ sender, System::EventArgs^ e) // обработчик нажатия кнопки сохранения рисунка

{

String^ filename;

saveFileDialog1->ShowDialog(); // открываем окно для сохранения

filename = saveFileDialog1->FileName; // сохраняем название изображения

if (filename == "") // если окно было закрыто ничего не делаем

return;

Bitmap^ image = gcnew Bitmap(pictureBox1->Image); // создаём сохраняемый объект, присваиваем ему текущее изображение

image->Save(filename); // сохраняем

MessageBox::Show("Изображение сохранено!", "Успешно.");

return System::Void();

}

};

}

Пример работы программы.



Рисунок 1. Окно программы.

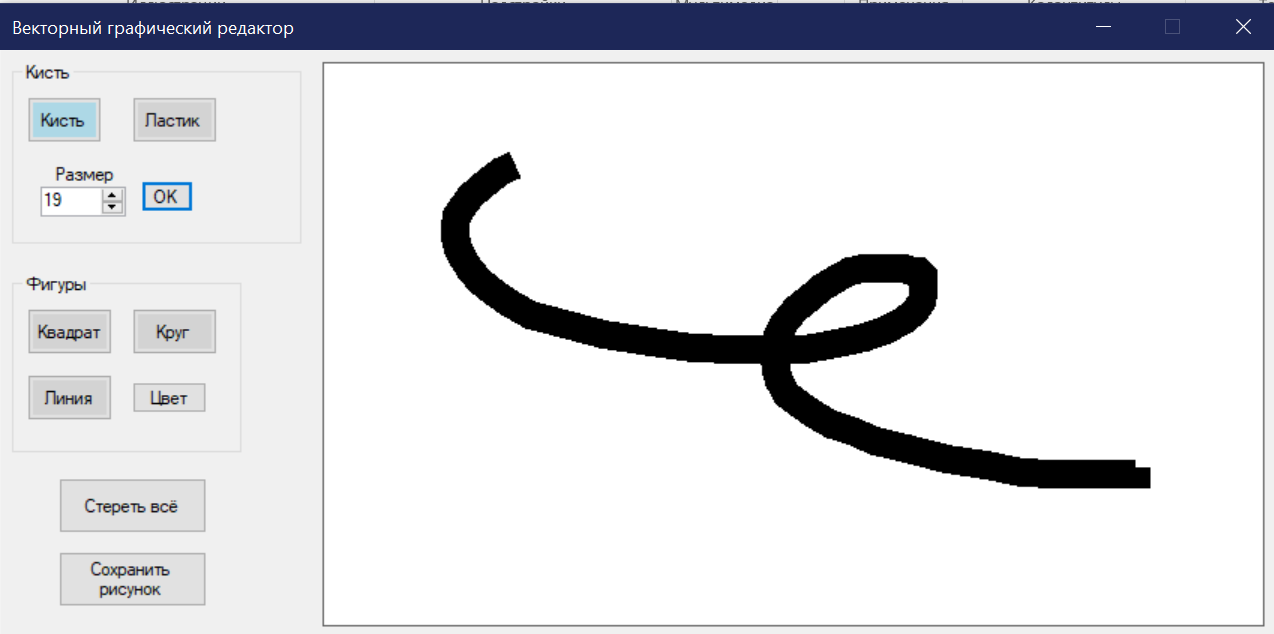


Рисунок 2. Рисунок кистью.

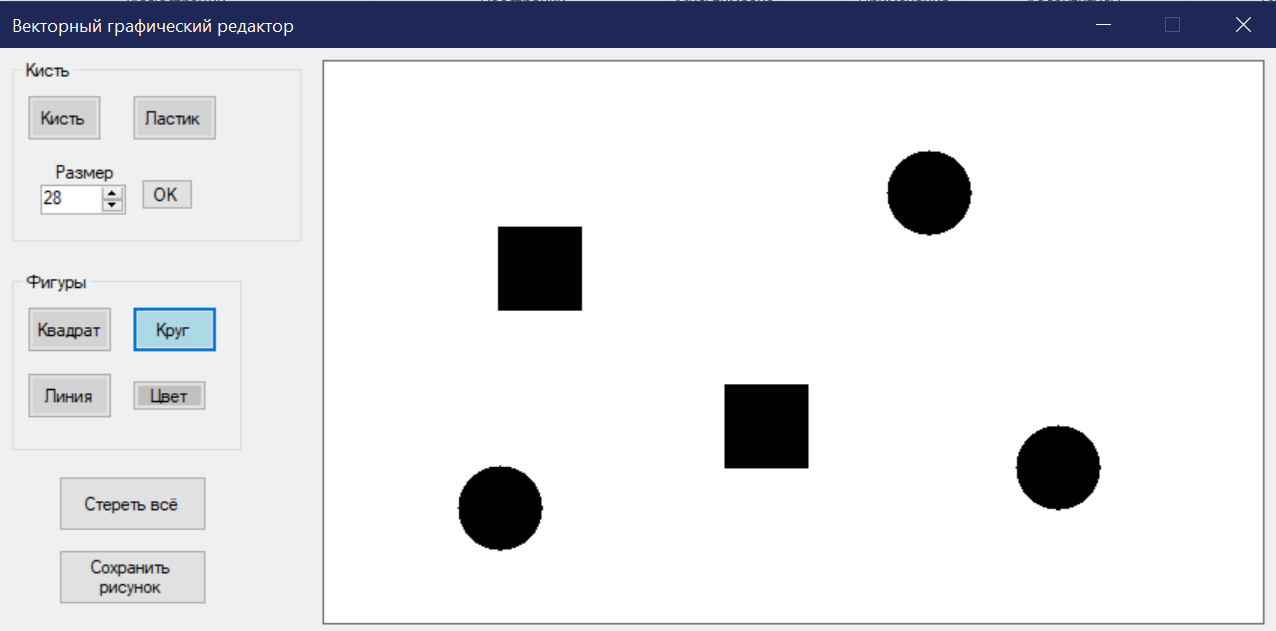


Рисунок 3. Рисунок геометрическими фигурами.

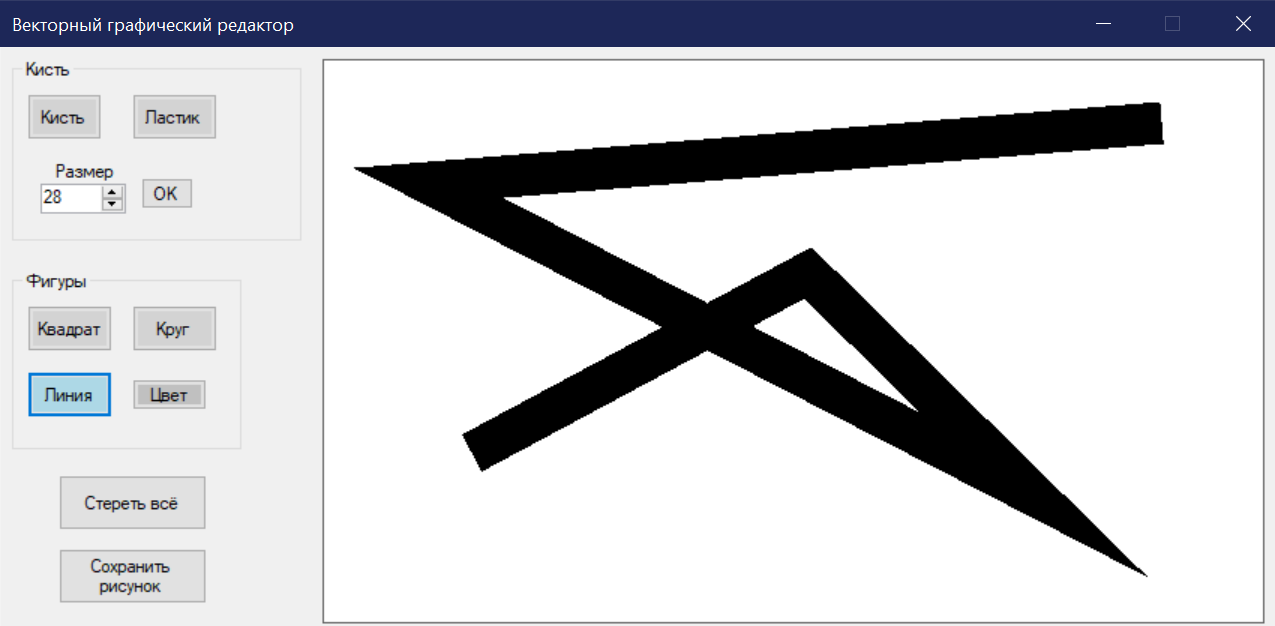


Рисунок 4. Рисунок линией.

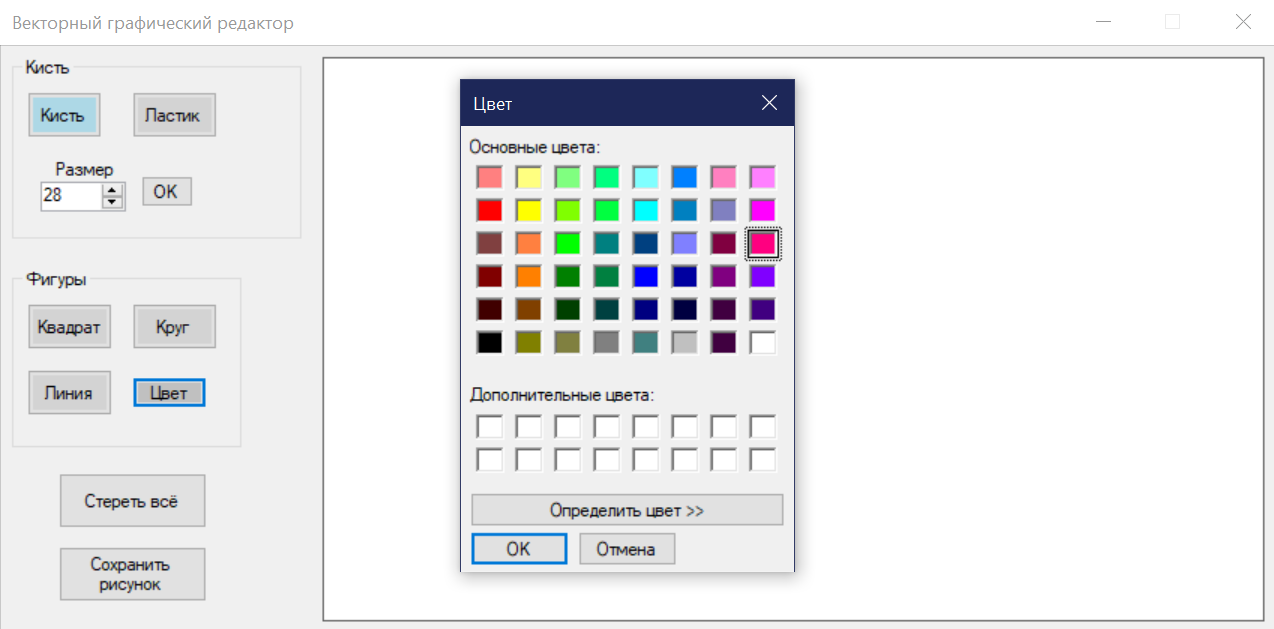


Рисунок 5. Нажав на кнопку «Цвет» можно выбрать цвет кисти.



Рисунок 6. Новый цвет.

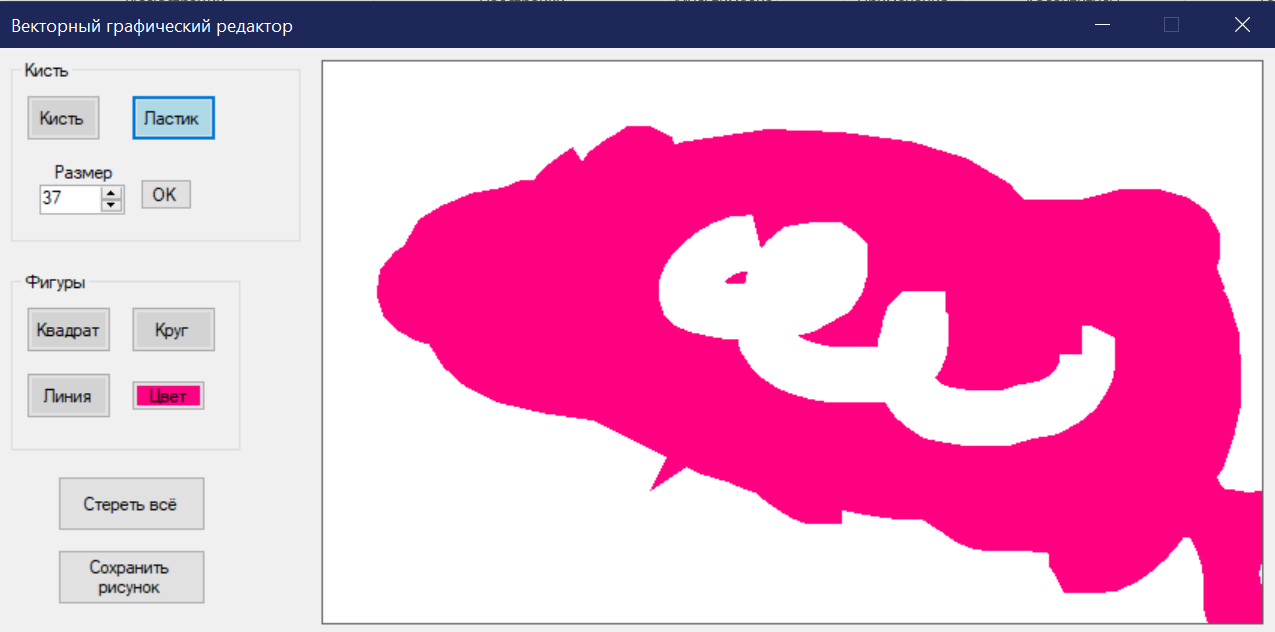


Рисунок 7. Кисть «Ластик».

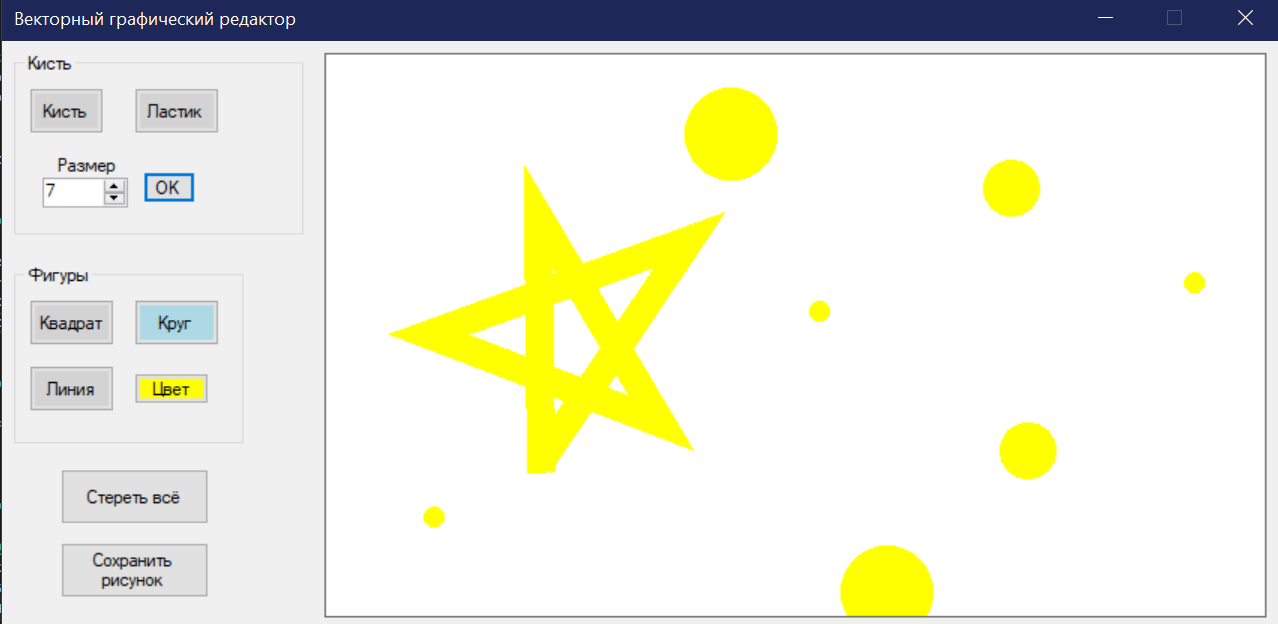


Рисунок 8. Кнопка «сохранить рисунок»

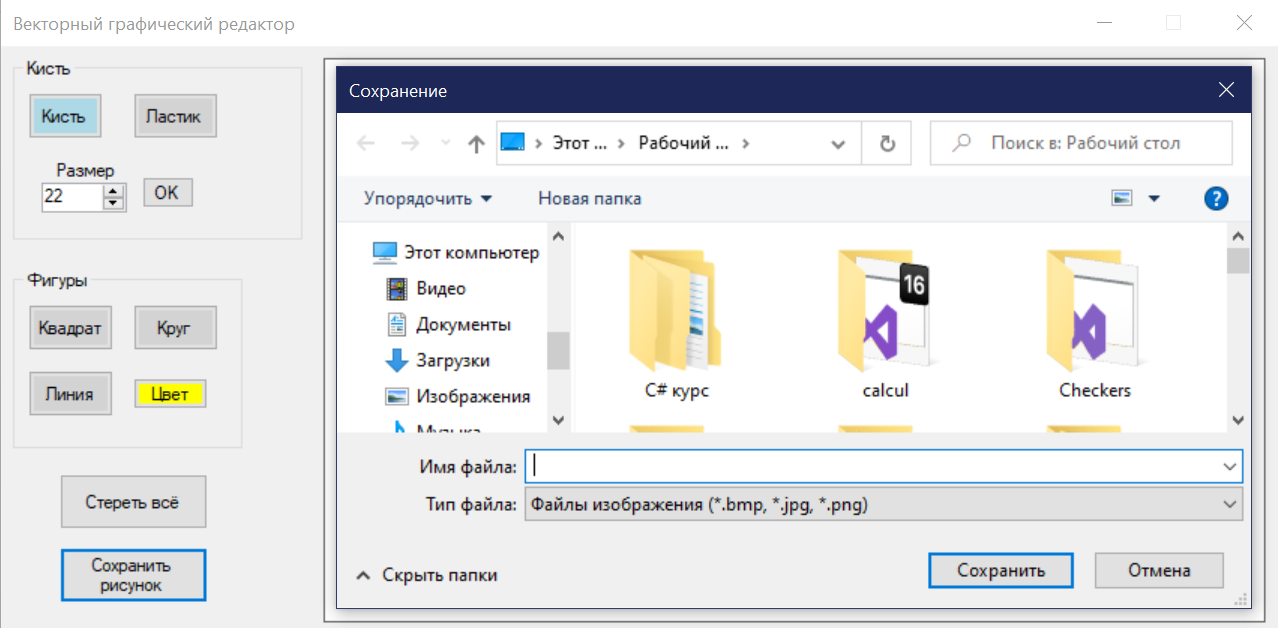


Рисунок 9. Сохранение изображения.

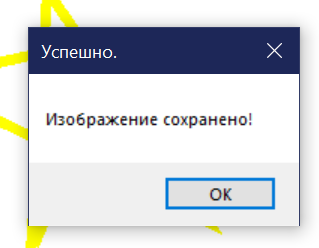


Рисунок 10. Сообщение: успешно.

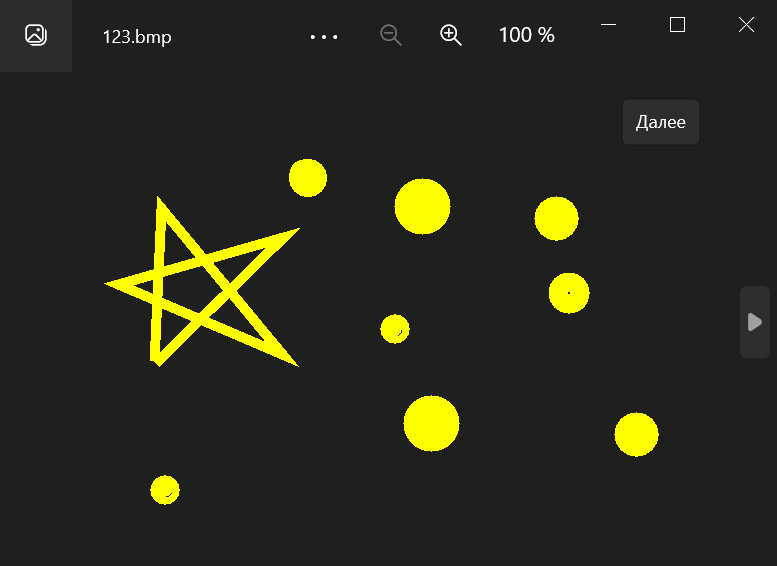


Рисунок 11. Изображение.

Вывод.

В данном индивидуальном задании я закрепила навыки ООП на примере реализации векторного графического редактора.

Контрольные вопросы.

1. **Зачем нужен перегруженный оператор присваивания?**

Перегруженный оператор присваивания используется для присваивания объектов одного конкретного класса друг другу.

1. **Зачем нужен механизм наследования?**

Наследование – это механизм создания нового класса на основе уже существующего. Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

1. **Зачем используются модификаторы при наследовании классов? Какие это модификаторы?**

Существуют модификаторы доступа и наследования - public, private, protected.  
В модификаторах доступа:

Public — доступ открыт всем, кто видит определение данного класса.

Protected — доступ открыт классам, производным от данного. То есть, производные классы получают свободный доступ к таким свойствам или метода. Все другие классы такого доступа не имеют.

Private — доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса - как функциям, так и классам. Однако производные классы не получают доступа к этим данным совсем. И все другие классы такого доступа не имеют.

В C++ существует public-наследование, private-наследование и protected-наследование. В зависимости от того, какой тип используется, изменяется доступ к членам базового класса для клиентов производного.

1. **Зачем нужен механизм полиморфизма?**

Полиморфизм - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. В С++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций.

В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий.

1. **Что понимается под динамическим полиморфизме?**

Динамический полиморфизм предстает перед нами в форме классов с виртуальными функциями и объектов, работа с которыми осуществляется косвенно — через указатели или ссылки, в то время как статический полиморфизм включает шаблоны классов и функций.

1. **Что такое интерфейс класса?**

Интерфейс – это класс, который не имеет переменных-членов, и все методы которого являются чистыми виртуальными функциями.

1. **Зачем нужен чисто виртуальный метод?**

Чисто виртуальный метод является методом, который объявляется в базовом классе, но не имеет в нем определения. Поскольку он не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию определения. Для объявления чисто виртуальной функции используется следующая общая форма:

virtual тип имя\_функции(список параметров) = 0;

Здесь тип обозначает тип возвращаемого значения, а имя\_функции является именем функции.

При введении чисто виртуальной функции в производном классе обязательно необходимо опре­делить свою собственную реализацию этой функции. Если класс не будет содержать определения этой функции, то компилятор выдаст ошибку.

1. **Какой класс называется абстрактным?**

Если какой-либо класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется абстрактным. Абстрактный класс служит в качестве базового для других производных классов.