**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизации систем управления

КУРСОВАЯ РАБОТА

по программированию

Разработка графического редактора на языке С++ с использованием механизмов ООП

Студент Марков Д.Ю. Группа АИ-21-2

Руководитель Кургасов В.В.

Доцент

Липецк 2022 г.

Задание кафедры

Реализовать на языке С++ векторный графический редактор. В ходе выполнения работы обязательно применение объектно-ориентированных возможностей языка С++: 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 С++.

Аннотация

С. 40. Ил. 7. Литература 3 назв. Прил. 1;

В работе описывается векторный графический редактор, позволяющий создавать, редактировать и сохранять векторные изображения. Доступны для рисования три вида фигур: прямоугольник, эллипс и линия.

Оглавление

[1 Введение 5](#_Toc57549164)

[2 Диаграмма классов 6](#_Toc57549165)

[3 Описание программы 7](#_Toc57549166)

[3.1 Общие сведения 7](#_Toc57549167)

[3.2 Функциональное назначение 7](#_Toc57549168)

[3.3 Описание логической структуры 7](#_Toc57549169)

[3.3.1 Общее описание 7](#_Toc57549170)

[3.3.2 Открытие файла 8](#_Toc57549171)

[3.3.3 Сохранение файла 8](#_Toc57549172)

[3.3.4 Добавление фигур 8](#_Toc57549173)

[3.3.5 Перемещение фигур 9](#_Toc57549174)

[3.3.6 Удаление фигуры 10](#_Toc57549175)

[3.3.7 Очистка области рисования 10](#_Toc57549176)

[3.4 Используемые технические средства 10](#_Toc57549177)

[3.5 Вызов и загрузка 11](#_Toc57549178)

[3.6 Входные данные 11](#_Toc57549179)

[4 Описание применения 12](#_Toc57549180)

[4.1 Назначение программы 12](#_Toc57549181)

[4.2 Условия применения 12](#_Toc57549182)

[4.3 Описание задачи 12](#_Toc57549183)

[4.4 Входные и выходные данные 12](#_Toc57549184)

[5 Руководство оператора 14](#_Toc57549185)

[5.1 Сведения о форме представления программы 14](#_Toc57549186)

[5.2 Назначение и условия применения программ 14](#_Toc57549187)

[5.3 Характеристика программы 14](#_Toc57549188)

[5.4 Обращение к программе 15](#_Toc57549189)

[5.5 Входные и выходные данные 15](#_Toc57549190)

[5.6 Сообщения 15](#_Toc57549191)

[6 Описание контрольного примера 16](#_Toc57549192)

[7 Заключение 18](#_Toc57549193)

[8 Список источников 19](#_Toc57549194)

[Приложение А 20](#_Toc57549195)

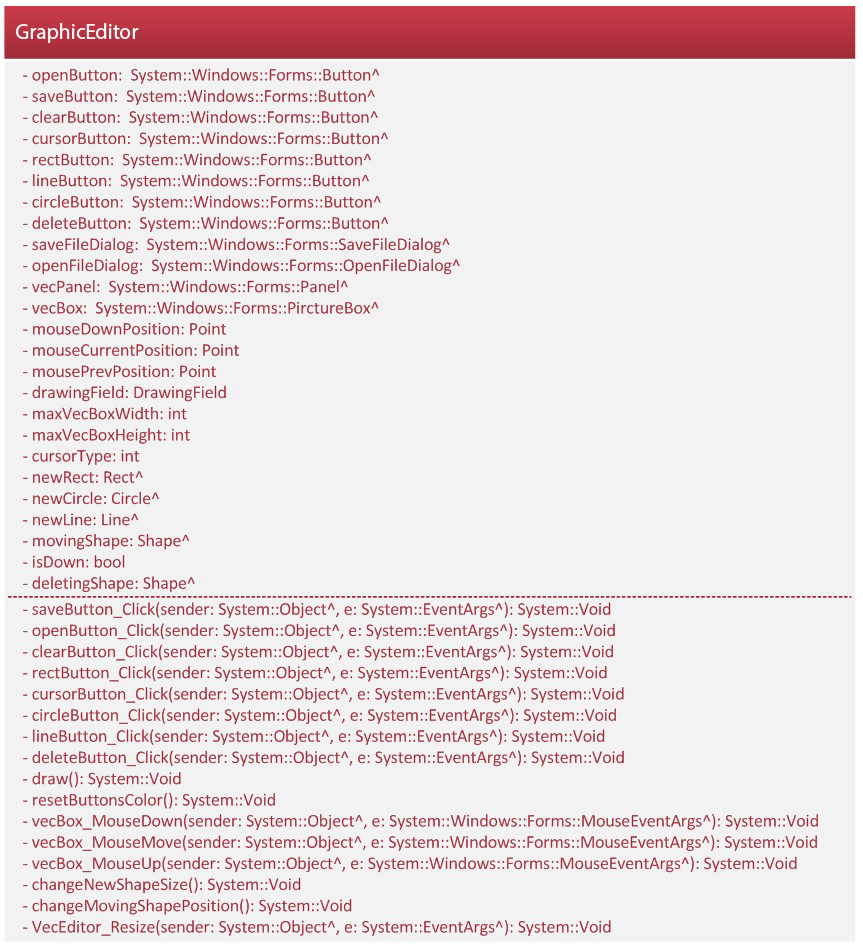
1 Введение

Векторный графический редактор – это программа, позволяющая пользователю создавать и редактировать векторные изображения непосредственно на экране компьютера, а также сохранять их.

Векторная графика – это способ представления объектов и изображений, основанный на математическом описании элементарных геометрических объектов.

В этой работе реализован векторный графический редактор. Для рисования доступны три вида фигур: прямоугольник, эллипс и линия. Так же имеется возможность перемещать уже нарисованные фигуры с помощью инструмента «курсор» и удалять нарисованные фигуры с помощью инструмента «удалить». Сохраняются изображения в файл с расширением «xml».

2 Диаграмма классов

 Рисунок 1 – Диаграмма классов

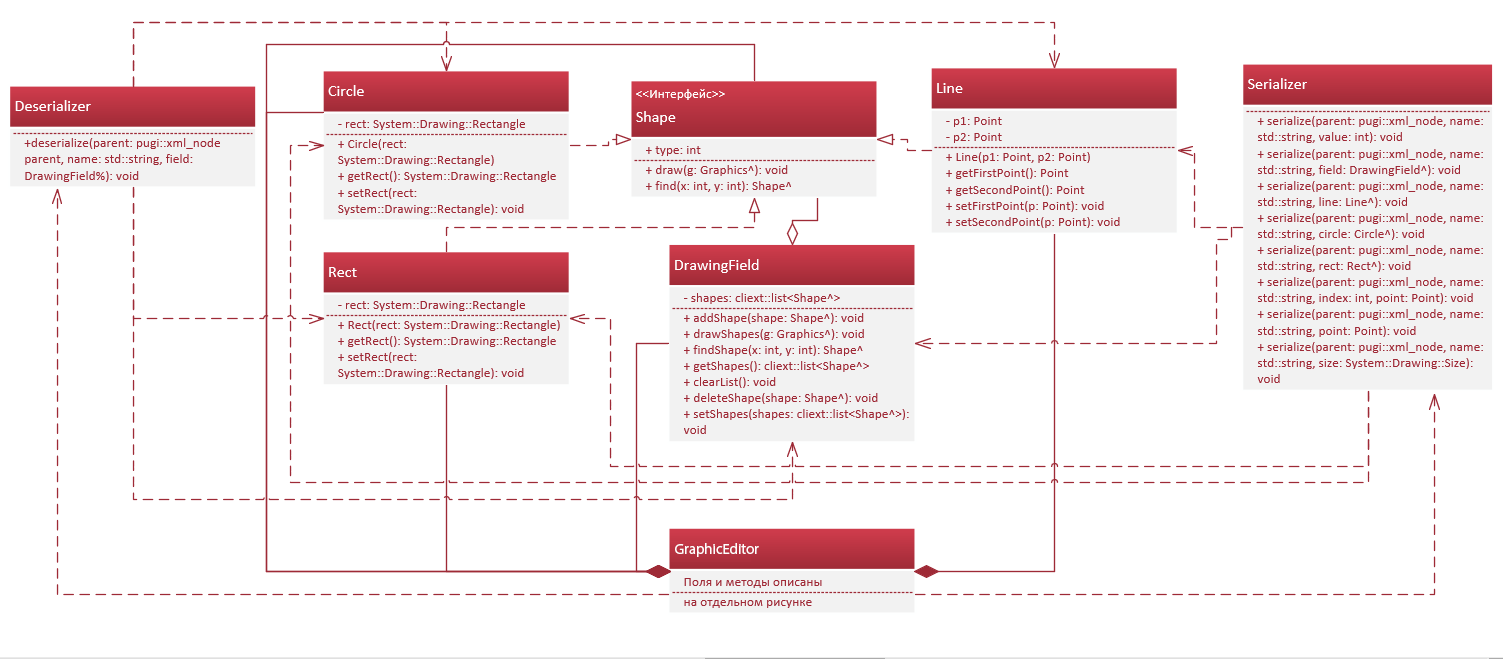


Рисунок 2 – диаграмма класса GraphicEditor

3 Описание программы

3.1 Общие сведения

Векторный графический редактор «GraphicEditor». Для функционирования программы необходима программная платформа .NET Framework версии 4.5 или выше. Программа написана на языке C++/CLI.

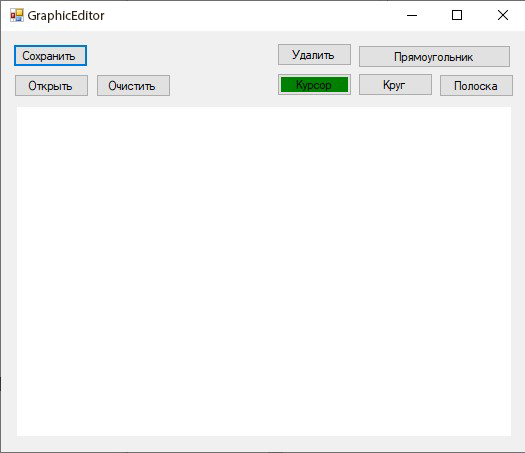


Рисунок 3 – Окно программы

3.2 Функциональное назначение

Программа предназначена для создания и (или) редактирования векторных изображений.

Доступные виды графических примитивов: прямоугольник, эллипс, линия.

3.3 Описание логической структуры

3.3.1 Общее описание

Отрисовка графического интерфейса и обработка пользовательского ввода осуществляется классом GraphicEditor.

Класс DrawingField содержит список добавленных фигур, позволяет добавлять, удалять, выводить и искать по заданным координатам добавленные фигуры.

Классы Circle, Rect, Line описывают эллипс, прямоугольник и линию соответственно. Все они наследуются от класса Shape и реализуют его методы draw() и find(int x, int y).

Класс Serializer позволяет сохранять изображение в файл с расширением «.xml».

Класс Deserializer позволяет загружать сохраненные в файл фигуры.

3.3.2 Открытие файла

При нажатии на кнопку «Открыть» вызывается окно выбора файла с помощью метода ShowDialog() элемента openFileDialog. При успешном выборе файла происходит считывание файла с помощью объекта класса pugi::xml\_document и добавление фигур из файла в экземпляр класса DrawingField с помощью экземпляра класса Deserializer.

3.3.3 Сохранение файла

При нажатии на кнопку «Сохранить» у пользователя запрашивается местоположение для сохранения файла с помощью метода ShowDialog() элемента saveFileDialog. При успешном выборе местоположения происходит запись в файл с помощью экземпляров классов pugi::xml\_document и Serializer.

3.3.4 Добавление фигур

Для добавления фигур используются следующие обработчики событий: vecBox\_MouseDown – нажатие ЛКМ в пределах области рисования, vecBox\_MouseMove – перемещение указателя в пределах области рисования, vecBox\_MouseUp – отпускание ЛКМ в пределах области рисования, и методы: draw() – вывод в область рисования добавленных фигур, changeNewShapeSize – для изменения размера добавляемой фигуры, addShape(Shape^) класса DrawingField – для помещения нового элемента в список добавленных фигур. Тип добавляемой фигуры зависит от значения поля cursorType: 1 – линия, 2 – прямоугольник, 3 – эллипс. Значение поля cursorType изменяется при нажатии на кнопки «Прямоугольник», «Эллипс», «Линия», «Курсор», «Удалить».

При срабатывании обработчика событий vecBox\_MouseDown полю isDown задается значение true, положение указателя помещается в поля mouseDownPosition и mousePrevPosition, выделяется память для экземпляра соответствующего класса (в зависимости от значения поля cursorType) и вызывается метод draw() для вывода добавляемой фигуры в область рисования.

При срабатывании обработчика событий vecBox\_MouseMove положение указателя помещается в поле mouseCurrentPosition, вызывается метод changeNewShapeSize(), изменяющий координаты фигуры, и добавляемая фигура выводится в область рисования.

При срабатывании обработчика событий vecBox\_MouseUp новый элемент добавляется в список фигур с помощью метода addShape() экземпляра класса DrawingField.

3.3.5 Перемещение фигур

Для перемещения фигур используются те же обработчики событий и методы, что и для добавления фигур, кроме changeNewShapeSize() – вместо него вызывается changeMovingShapePosition(). Значение поля cursorType при этом должно быть 0.

При срабатывании обработчика событий vecBox\_MouseDown полю isDown задается значение true, положение указателя помещается в поля mouseDownPosition и mousePrevPosition, в поле movingShape помещается результат выполнения метода findShape(int x, int y) экземпляра класса DrawingField. Этот метод возвращает ссылку на элемент, находящийся в заданных координатах, и удаляет из списка добавленных фигур найденную. Если фигур в заданных координатах не нашлось, возвращается nullptr.

При срабатывании обработчика событий vecBox\_MouseMove координаты указателя помещаются в поле mouseCurrentPosition, проверяется, равно ли значение поля movingShape nullptr. Если не равно, вызывается метод changeMovingShapePosition(), который изменяет координаты перемещаемой фигуры, полю mousePrevPosition присваивается значение поля mouseCurrentPosition и в область рисования выводятся добавленный и перемещаемая фигуры.

При срабатывании обработчика событий vecBox\_MouseUp проверяется значение поля movingShape. Если оно не равно nullptr, то вызывается метод addShape(movingShape) экземпляра класса DrawingField.

3.3.6 Удаление фигуры

Для удаления фигуры используется обработчик событий vecBox\_MouseDown. При этом значение поля cursorType должно быть 4.

При срабатывании обработчика событий vecBox\_MouseDown полю deletingShape присваивается результат выполнения метода findShape(x, y) экземпляра класса DrawingFiled. Если значение не равно nullprt, то найденная фигура удаляется из списка добавленных.

3.3.7 Очистка области рисования

При нажатии на кнопку «Очистить» вызывается метод clearList() у экземпляра класса DrawingField. Поле рисования заливается белым цветом.

3.4 Используемые технические средства

Минимальные требования для работы программы:

1. ОС Windows Vista, 8, 8.1 или 10;
2. Платформа Microsoft .NET Framework версии не ниже 4.5
3. Процессор, совместимый с ОС Windows Vista, 8, 8.1 или 10;
4. Не менее 5 МБ свободной оперативной памяти
5. Не менее 254 КБ свободной памяти на ЖД
6. Мышь
7. Монитор

3.5 Вызов и загрузка

Для вызова программы используется исполняемый файл Kurs.exe. Его необходимо запустить с помощью файлового менеджера или командной строки. После запуска программа использует 5 МБ оперативной памяти. При добавлении фигур объем используемой оперативной памяти увеличивается.

3.6 Входные данные

Первым видом входных данных являются координаты указателя и выбранный тип указателя («Курсор», «Удалить», «Прямоугольник», «Эллипс», «Линия»). Тип указателя изменяется нажатием на одноименные кнопки. Выбранная в текущий момент кнопка закрашена зеленым цветом.

Второй вид входных данных – файл с расширением «.xml» предварительно созданный с помощью этой программы.

3.7 Выходные данные

Выходными данными является изображение в области рисования, которое можно сохранить в файл с расширением «.xml».

4 Описание применения

4.1 Назначение программы

Программа применяется для создания и редактирования векторных изображений. Доступны три вида графических примитивов: эллипс, прямоугольник, линия.

4.2 Условия применения

Для применения программы необходим персональный компьютер, удовлетворяющий минимальным требованиям:

1. ОС Windows Vista, 8, 8.1 или 10;
2. Платформа Microsoft .NET Framework версии не ниже 4.5
3. Процессор, совместимый с ОС Windows Vista, 8, 8.1 или 10;
4. Не менее 5 МБ свободной оперативной памяти
5. Не менее 254 КБ свободной памяти на ЖД
6. Мышь
7. Монитор

4.3 Описание задачи

Задача программы – создание и редактирование векторных изображений. Новое изображение можно нарисовать с помощью фигур «Эллипс», «Прямоугольник», «Линия». Чтобы удалить фигуру из области рисования нажмите на кнопку «Удалить», затем нажмите ЛКМ на удаляемую фигуру. Для перемещения фигуры нажмите на кнопку «Курсор», после чего нажмите ЛКМ на нужную фигуру и переместите ее. Для сохранения используется кнопка «Сохранить», для открытия существующего изображения – кнопка «Открыть».

4.4 Входные и выходные данные

Входными данными может быть уже созданный с помощью данной программы файл с изображением, либо данные введенные с помощью мыши: координаты нажатия на область рисования и перемещение указателя в области рисования.

Выходными данными является изображение в области рисование, которое можно сохранить в файл с расширением «.xml».

5 Руководство оператора

5.1 Сведения о форме представления программы

Программа записана на USB-накопитель с файловой системой «FAT32». Для чтения информации с носителя необходимо устройство, способное считывать информацию из USB-накопителя и поддерживающее работу с файловой системой «FAT32».

5.2 Назначение и условия применения программ

Программа предназначена для создания и редактирования векторных изображений. Доступны следующие функции: открытие изображения, добавление в область рисования фигур «Прямоугольник», «Эллипс» и «Линия», перемещение добавленных фигур, удаление добавленных фигур, очистка области рисования и сохранение изображения.

Минимальные требования для применения программы:

1. ОС Windows Vista, 8, 8.1 или 10;
2. Платформа Microsoft .NET Framework версии не ниже 4.5
3. Процессор, совместимый с ОС Windows Vista, 8, 8.1 или 10;
4. Не менее 5 МБ свободной оперативной памяти
5. Не менее 254 КБ свободной памяти на ЖД
6. Мышь
7. Монитор

5.3 Характеристика программы

Скорость работы программы на примере открытия изображения с разным количеством фигур:

1. Windows 10, процессор Intel(R) Core(TM) i3-4160, встроенный компилятор среды разработки MS Visual Studio 16.8.2, платформа .NET Framework версии 4.8.04084 размер входных данных: 50 фигур типа Rect; время с момента выбора файла до отображения фигур: 15 мс.
2. Windows 10, процессор Intel(R) Core(TM) i3-4160, встроенный компилятор среды разработки MS Visual Studio 16.8.2, платформа .NET Framework версии 4.8.04084, размер входных данных: 5000 фигур типа Rect; время с момента выбора файла до отображения фигур: 89 мс.

5.4 Обращение к программе

Программа вызывается, после её компилирования, через командную строку или файловый менеджер. Данные в программу вводятся пользователем с помощью мыши.

5.5 Входные и выходные данные

Входные данные – это координаты указателя в области рисования, они принимаются в качестве аргумента соответствующих обработчиков событий, или файл с изображением.

Выходные данные – это изображение в области рисования, которое можно сохранить в файл с расширением «.xml».

5.6 Сообщения

1. При нажатии на кнопку «Сохранить» появляется окно с заголовком «Сохранить изображение». Необходимо выбрать местоположение для сохранения и имя файла. После чего нажать на кнопку «Сохранить».

2. Если пользователь введет имя файла, совпадающее с уже существующим файлом, появится диалоговое окно с сообщением «”название файла” уже существует. Вы хотите заменить его?». Если совпадающее имя выбрано намеренно, нажмите на кнопку «Да», иначе нажмите на кнопку «Нет» и введите другое имя файла.

3. При нажатии на кнопку «Открыть» появляется окно с заголовком «Открыть изображение». Необходимо выбрать файл с изображением, затем нажать на кнопку «Открыть».

6 Описание контрольного примера

Запустим программу.

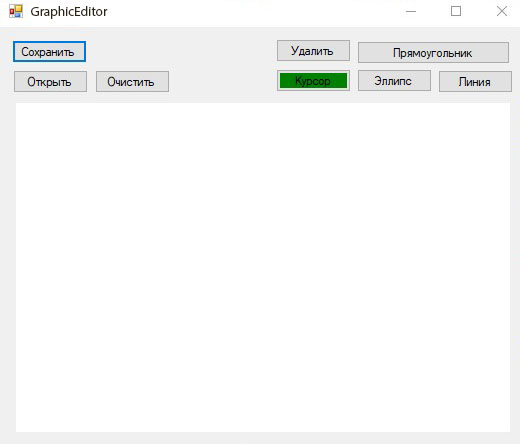


Рисунок 4 – Окно программы после запуска

Выберем тип фигуры «Прямоугольник», нажав на одноименную кнопку, и добавим один экземпляр в область рисования.

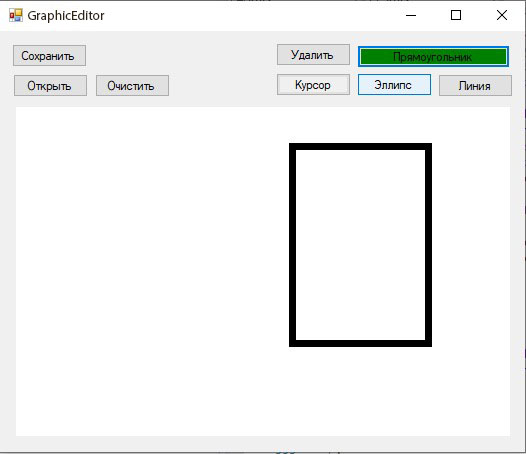


Рисунок 5 – В область рисования добавлена фигура «Прямоугольник»

Выберем тип фигуры «Линия», нажав на одноименную кнопку, и добавим два экземпляра в область рисования.

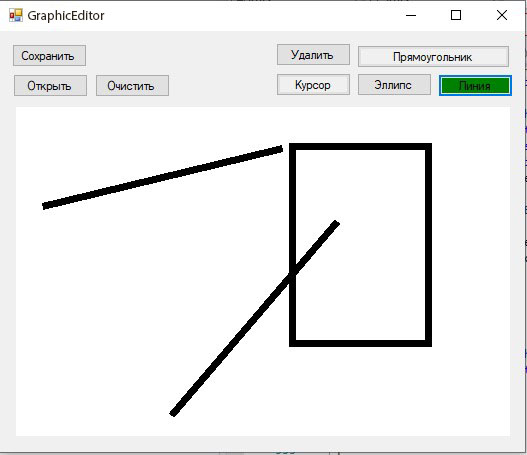


Рисунок 6 – Добавлены две фигуры типа «Линия»

Переместим фигуру типа «Прямоугольник», выбрав тип указателя «Курсор».

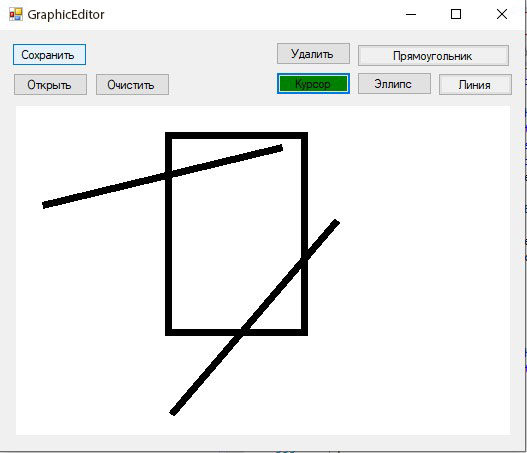


Рисунок 7 – Перемещена фигура типа «Прямоугольник»

7 Заключение

Векторный редактор «GraphicEditor» предоставляет основные возможности для редактирования векторных изображений: добавление фигур трех видов («Прямоугольник», «Эллипс», «Линия»), перемещение фигур, удаление фигур, сохранение изображение, открытие изображения.

Возможная область применения программы – создание изображений из простых геометрических фигур.

8 Список источников

1. ГОСТ 19.504 – 79. Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению. – Москва: Изд-во Стандартинформ, 2010. – 2 с.
2. ГОСТ 19.502 – 78. Единая система программной документации. Описание применения. Требования к содержанию и оформлению. – Москва: Изд-во Стандартинформ, 2010. – 2 с.
3. ГОСТ 19.402 – 78. Единая система программной документации. Описание программы. – Москва: Изд-во Стандартинформ, 2010. – 2 с.

Приложение А

(обязательное)

Содержание файла GraphicEditor.cpp:

#include "GraphicEditor.h"

#include <Windows.h>

using namespace Kurs;

[STAThreadAttribute]

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew GraphicEditor);

return 0;

}

Содержание файла GraphicEditor.h:

#pragma once

#include "DrawingField.h"

#include "Circle.h"

#include "Line.h"

#include "Rect.h"

#include "Serializer.h"

#include <msclr\marshal\_cppstd.h>

#include "Deserializer.h"

#include "Serializer.h"

namespace Kurs {

constexpr auto CURSOR = 0;

constexpr auto LINE = 1;

constexpr auto RECTANGLE = 2;

constexpr auto CIRCLE = 3;

constexpr auto DEL = 4;

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для GraphicEditor

/// </summary>

public ref class GraphicEditor : public System::Windows::Forms::Form

{

public:

GraphicEditor(void)

{

InitializeComponent();

vecPanel->AutoScroll = true;

vecBox->SizeMode = PictureBoxSizeMode::AutoSize;

maxVecBoxHeight = vecBox->Height;

maxVecBoxWidth = vecBox->Width;

isDown = false;

Bitmap^ bitmap = gcnew Bitmap(vecBox->Size.Width, vecBox->Size.Height);

Graphics^ g = Graphics::FromImage(bitmap);

g->FillRectangle(gcnew SolidBrush(Color::White), 0, 0, vecBox->Width, vecBox->Height);

vecBox->Image = bitmap;

draw();

cursorType = CURSOR;

cursorButton->BackColor = Color::Green;

}

protected:

~GraphicEditor()

{

if (components)

{

delete components;

}

}

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

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

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

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

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

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

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

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

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

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

private: System::Windows::Forms::Panel^ vecPanel;

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

protected:

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void)

{

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

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

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

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

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

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

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

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

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

this->vecPanel = (gcnew System::Windows::Forms::Panel());

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

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

this->vecPanel->SuspendLayout();

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

this->SuspendLayout();

//

// saveButton

//

this->saveButton->Location = System::Drawing::Point(12, 13);

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

this->saveButton->Size = System::Drawing::Size(75, 23);

this->saveButton->TabIndex = 1;

this->saveButton->Text = L"Сохранить";

this->saveButton->UseVisualStyleBackColor = true;

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

//

// openButton

//

this->openButton->Location = System::Drawing::Point(13, 43);

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

this->openButton->Size = System::Drawing::Size(75, 23);

this->openButton->TabIndex = 2;

this->openButton->Text = L"Открыть";

this->openButton->UseVisualStyleBackColor = true;

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

//

// clearButton

//

this->clearButton->Location = System::Drawing::Point(95, 43);

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

this->clearButton->Size = System::Drawing::Size(75, 23);

this->clearButton->TabIndex = 3;

this->clearButton->Text = L"Очистить";

this->clearButton->UseVisualStyleBackColor = true;

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

//

// cursorButton

//

this->cursorButton->Location = System::Drawing::Point(276, 42);

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

this->cursorButton->Size = System::Drawing::Size(75, 23);

this->cursorButton->TabIndex = 4;

this->cursorButton->Text = L"Курсор";

this->cursorButton->UseVisualStyleBackColor = true;

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

//

// lineButton

//

this->lineButton->Location = System::Drawing::Point(438, 43);

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

this->lineButton->Size = System::Drawing::Size(75, 23);

this->lineButton->TabIndex = 5;

this->lineButton->Text = L"Линия";

this->lineButton->UseVisualStyleBackColor = true;

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

//

// rectButton

//

this->rectButton->Location = System::Drawing::Point(357, 14);

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

this->rectButton->Size = System::Drawing::Size(153, 23);

this->rectButton->TabIndex = 6;

this->rectButton->Text = L"Прямоугольник";

this->rectButton->UseVisualStyleBackColor = true;

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

//

// circleButton

//

this->circleButton->Location = System::Drawing::Point(357, 42);

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

this->circleButton->Size = System::Drawing::Size(75, 23);

this->circleButton->TabIndex = 7;

this->circleButton->Text = L"Эллипс";

this->circleButton->UseVisualStyleBackColor = true;

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

//

// openFileDialog1

//

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

//

// vecPanel

//

this->vecPanel->Controls->Add(this->vecBox);

this->vecPanel->Location = System::Drawing::Point(12, 72);

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

this->vecPanel->Size = System::Drawing::Size(501, 337);

this->vecPanel->TabIndex = 8;

//

// vecBox

//

this->vecBox->Location = System::Drawing::Point(4, 4);

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

this->vecBox->Size = System::Drawing::Size(494, 329);

this->vecBox->TabIndex = 0;

this->vecBox->TabStop = false;

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

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

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

//

// deleteButton

//

this->deleteButton->Location = System::Drawing::Point(276, 12);

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

this->deleteButton->Size = System::Drawing::Size(75, 23);

this->deleteButton->TabIndex = 9;

this->deleteButton->Text = L"Удалить";

this->deleteButton->UseVisualStyleBackColor = true;

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

//

// GraphicEditor

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

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

this->ClientSize = System::Drawing::Size(525, 421);

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

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

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

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

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

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

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

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

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

this->MinimumSize = System::Drawing::Size(541, 460);

this->Name = L"GraphicEditor";

this->Text = L"GraphicEditor";

this->Resize += gcnew System::EventHandler(this, &GraphicEditor::GraphicEditor\_Resize);

this->vecPanel->ResumeLayout(false);

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

this->ResumeLayout(false);

}

#pragma endregion

private: Point mouseDownPosition;

private: Point mouseCurrentPosition;

private: Point mousePrevPosition;

private: DrawingField drawingField;

private: int maxVecBoxWidth;

private: int maxVecBoxHeight;

private: int cursorType;

private: Rect^ newRect;

private: Circle^ newCircle;

private: Line^ newLine;

private: Shape^ movingShape;

private: bool isDown;

private: Shape^ deletingShape;

private: System::Void saveButton\_Click(System::Object^ sender, System::EventArgs^ e) {

saveFileDialog1->Title = "Сохранить изображение";

saveFileDialog1->Filter = "XML File (\*.xml) | \*.xml";

if(saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)

{

pugi::xml\_document doc;

Serializer serializer;

serializer.serialize(doc.root(), NAME\_OF(DrawingField), %drawingField);

doc.save\_file(msclr::interop::marshal\_as<std::string>(saveFileDialog1->FileName).c\_str());

}

}

private: System::Void openButton\_Click(System::Object^ sender, System::EventArgs^ e) {

openFileDialog1->Title = "Открыть изображение";

openFileDialog1->Filter = "XML File (\*.xml) | \*.xml";

if(openFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK)

{

pugi::xml\_document doc;

doc.load\_file(msclr::interop::marshal\_as<std::string>(openFileDialog1->FileName).c\_str());

Deserializer deserializer;

deserializer.deserialize(doc.root(), NAME\_OF(DrawingField), drawingField);

draw();

}

}

private: System::Void clearButton\_Click(System::Object^ sender, System::EventArgs^ e) {

drawingField.clearList();

draw();

}

private: System::Void rectButton\_Click(System::Object^ sender, System::EventArgs^ e) {

resetButtonsColor();

rectButton->BackColor = Color::Green;

cursorType = RECTANGLE;

}

private: System::Void cursorButton\_Click(System::Object^ sender, System::EventArgs^ e) {

resetButtonsColor();

cursorButton->BackColor = Color::Green;

cursorType = CURSOR;

}

private: System::Void circleButton\_Click(System::Object^ sender, System::EventArgs^ e) {

resetButtonsColor();

circleButton->BackColor = Color::Green;

cursorType = CIRCLE;

}

private: System::Void lineButton\_Click(System::Object^ sender, System::EventArgs^ e) {

resetButtonsColor();

lineButton->BackColor = Color::Green;

cursorType = LINE;

}

private: System::Void deleteButton\_Click(System::Object^ sender, System::EventArgs^ e) {

resetButtonsColor();

deleteButton->BackColor = Color::Green;

cursorType = DEL;

}

private: System::Void draw() {

Graphics^ g = Graphics::FromImage(vecBox->Image);

g->FillRectangle(gcnew SolidBrush(Color::White), 0, 0, vecBox->Width, vecBox->Height);

drawingField.drawShapes(g);

vecBox->Invalidate();

}

private: System::Void resetButtonsColor()

{

switch (cursorType)

{

case CURSOR:

cursorButton->BackColor = Control::BackColor;

break;

case RECTANGLE:

rectButton->BackColor = Control::BackColor;

break;

case CIRCLE:

circleButton->BackColor = Control::BackColor;

break;

case LINE:

lineButton->BackColor = Control::BackColor;

break;

case DEL:

deleteButton->BackColor = Control::BackColor;

break;

}

}

private: System::Void vecBox\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

isDown = true;

mouseDownPosition = Point(e->X, e->Y);

mousePrevPosition = Point(e->X, e->Y);

draw();

switch (cursorType)

{

case CURSOR:

movingShape = drawingField.findShape(e->X, e->Y);

if(movingShape != nullptr)

{

switch(movingShape->type)

{

case LINE:

movingShape = (Line^)movingShape;

break;

case RECTANGLE:

movingShape = (Rect^)movingShape;

break;

case CIRCLE:

movingShape = (Circle^)movingShape;

break;

}

}

break;

case RECTANGLE:

newRect = gcnew Rect(System::Drawing::Rectangle(mouseDownPosition, System::Drawing::Size(1, 1)));

newRect->draw(Graphics::FromImage(vecBox->Image));

break;

case LINE:

newLine = gcnew Line(mouseDownPosition, mouseDownPosition);

newLine->draw(Graphics::FromImage(vecBox->Image));

break;

case CIRCLE:

newCircle = gcnew Circle(System::Drawing::Rectangle(mouseDownPosition, System::Drawing::Size(1, 1)));

newCircle->draw(Graphics::FromImage(vecBox->Image));

break;

case DEL:

deletingShape = drawingField.findShape(e->X, e->Y);

if (deletingShape != nullptr)

drawingField.deleteShape(deletingShape);

deletingShape = nullptr;

break;

}

vecBox->Invalidate();

}

private: System::Void vecBox\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

if(isDown)

{

mouseCurrentPosition = Point(e->X, e->Y);

if (movingShape == nullptr) {

changeNewShapeSize();

draw();

switch (cursorType)

{

case CURSOR:

break;

case RECTANGLE:

newRect->draw(Graphics::FromImage(vecBox->Image));

break;

case LINE:

newLine->draw(Graphics::FromImage(vecBox->Image));

break;

case CIRCLE:

newCircle->draw(Graphics::FromImage(vecBox->Image));

break;

}

} else

{

changeMovingShapePosition();

draw();

movingShape->draw(Graphics::FromImage(vecBox->Image));

mousePrevPosition = mouseCurrentPosition;

}

vecBox->Invalidate();

}

}

private: System::Void vecBox\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e) {

isDown = false;

switch (cursorType)

{

case CURSOR:

if (movingShape != nullptr) {

drawingField.addShape(movingShape);

movingShape = nullptr;

}

break;

case RECTANGLE:

drawingField.addShape(newRect);

break;

case LINE:

drawingField.addShape(newLine);

break;

case CIRCLE:

drawingField.addShape(newCircle);

break;

}

draw();

}

private: System::Void changeNewShapeSize()

{

int dX = mouseCurrentPosition.X - mouseDownPosition.X;

int dY = mouseCurrentPosition.Y - mouseDownPosition.Y;

switch (cursorType)

{

case CURSOR:

return;

case RECTANGLE: {

System::Drawing::Size s(fabs(mouseCurrentPosition.X - mouseDownPosition.X), fabs(mouseCurrentPosition.Y - mouseDownPosition.Y));

Point p(fmin(mouseCurrentPosition.X, mouseDownPosition.X), fmin(mouseCurrentPosition.Y, mouseDownPosition.Y));

if (p.X < 0 || p.X + s.Width > vecBox->Width || p.Y < 0 || p.Y + s.Height > vecBox->Height)

return;

newRect->setRect(System::Drawing::Rectangle(p, s));

break;

}

case LINE:

if (mouseCurrentPosition.X < 0 || mouseCurrentPosition.X > vecBox->Width || mouseCurrentPosition.Y < 0 || mouseCurrentPosition.Y > vecBox->Height)

return;

newLine->setFirstPoint(mouseDownPosition);

newLine->setSecondPoint(mouseCurrentPosition);

break;

case CIRCLE:

System::Drawing::Size s(fabs(mouseCurrentPosition.X - mouseDownPosition.X), fabs(mouseCurrentPosition.Y - mouseDownPosition.Y));

Point p(fmin(mouseCurrentPosition.X, mouseDownPosition.X), fmin(mouseCurrentPosition.Y, mouseDownPosition.Y));

if (p.X < 0 || p.X + s.Width > vecBox->Width || p.Y < 0 || p.Y + s.Height > vecBox->Height)

return;

newCircle ->setRect(System::Drawing::Rectangle(mouseDownPosition.X, mouseDownPosition.Y, dX, dY));

break;

}

}

private: System::Void changeMovingShapePosition()

{

System::Drawing::Rectangle r;

Point p1;

Point p2;

int dX = mouseCurrentPosition.X - mousePrevPosition.X;

int dY = mouseCurrentPosition.Y - mousePrevPosition.Y;

switch (movingShape->type)

{

case RECTANGLE:

r = ((Rect^)movingShape)->getRect();

if (r.X + dX < 0 || r.X + r.Width + dX > vecBox->Width || r.Y + dY < 0 || r.Y + r.Height + dY > vecBox->Height)

return;

((Rect^)movingShape)->setRect(System::Drawing::Rectangle(r.X + dX, r.Y + dY, r.Width, r.Height));

break;

case LINE:

p1 = ((Line^)movingShape)->getFirstPoint();

p2 = ((Line^)movingShape)->getSecondPoint();

if (p1.X + dX < 0 || p1.X + dX > vecBox->Width || p1.Y + dY < 0 || p1.Y + dY > vecBox->Height)

return;

if (p2.X + dX < 0 || p2.X + dX > vecBox->Width || p2.Y + dY < 0 || p2.Y + dY > vecBox->Height)

return;

((Line^)movingShape)->setFirstPoint(Point(p1.X + dX, p1.Y + dY));

((Line^)movingShape)->setSecondPoint(Point(p2.X + dX, p2.Y + dY));

break;

case CIRCLE:

r = ((Circle^)movingShape)->getRect();

if (r.Width < 0 && (r.X + dX > vecBox->Width || r.X + r.Width + dX < 0))

return;

else if (r.Width > 0 && (r.X + r.Width + dX > vecBox->Width || r.X + dX < 0))

return;

if (r.Height < 0 && (r.Y + dX > vecBox->Height || r.Y + r.Height + dX < 0))

return;

else if (r.Height > 0 && (r.Y + r.Height + dX > vecBox->Height || r.Y + dX < 0))

return;

((Circle^)movingShape)->setRect(System::Drawing::Rectangle(r.X + dX, r.Y + dY, r.Width, r.Height));

break;

}

}

private: System::Void GraphicEditor\_Resize(System::Object^ sender, System::EventArgs^ e) {

if (maxVecBoxWidth < this->Width || maxVecBoxHeight < this->Height)

{

vecPanel->Width = this->Width - 40;

vecPanel->Height = this->Height - 130;

vecBox->Width = this->Width - 47;

vecBox->Height = this->Height - 138;

Image^ oldImage = vecBox->Image;

Bitmap^ newBitmap = gcnew Bitmap(oldImage, vecBox->Width, vecBox->Height);

vecBox->Image = newBitmap;

maxVecBoxHeight = vecBox->Height;

maxVecBoxWidth = vecBox->Width;

draw();

}

else

{

vecPanel->Width = this->Width - 47;

vecPanel->Height = this->Height - 138;

}

}

};

}

Содержание файла Shape.h:

#pragma once

using namespace System::Drawing;

ref class Shape abstract

{

public:

int type;

virtual void draw(Graphics^ g) = 0;

virtual Shape^ find(int x, int y) = 0;

};

Содержание файла Rect.h:

#pragma once

#include "Shape.h"

ref class Rect :

public Shape

{

private:

System::Drawing::Rectangle rect;

public:

Rect(System::Drawing::Rectangle rect);

System::Drawing::Rectangle getRect();

void setRect(System::Drawing::Rectangle rect);

// Унаследовано через Shape

virtual void draw(Graphics^ g) override;

// Унаследовано через Shape

virtual Shape^ find(int x, int y) override;

};

Содержание файла Rect.cpp:

#include "Rect.h"

#include <cmath>

Rect::Rect(Rectangle rect)

{

this->rect = rect;

type = 2;

}

Rectangle Rect::getRect()

{

return rect;

}

void Rect::setRect(Rectangle rect)

{

this->rect = rect;

}

void Rect::draw(Graphics^ g)

{

g->DrawRectangle(gcnew Pen(Color::Black, 7), rect);

}

Shape^ Rect::find(int x, int y)

{

if (abs(rect.Top - y) <= 5 || abs(rect.Bottom - y) <= 5) {

if (rect.Left - 2 <= x && rect.Right + 2>= x)

return this;

} else if (abs(rect.Left - x) <= 7 || abs(rect.Right - x) <= 7)

{

if (rect.Bottom + 2>= y && rect.Top - 2 <= y)

return this;

}

return nullptr;

}

Содержание файла Line.h:

#pragma once

#include "Shape.h"

ref class Line :

public Shape

{

private:

Point p1;

Point p2;

public:

Line(Point p1, Point p2);

Point getFirstPoint();

Point getSecondPoint();

void setFirstPoint(Point p);

void setSecondPoint(Point p);

// Унаследовано через Shape

virtual void draw(Graphics^ g) override;

// Унаследовано через Shape

virtual Shape^ find(int x, int y) override;

};

Содержание файла Line.cpp:

#include "Line.h"

#include <cmath>

Line::Line(Point p1, Point p2)

{

this->p1 = p1;

this->p2 = p2;

type = 1;

}

Point Line::getFirstPoint()

{

return p1;

}

Point Line::getSecondPoint()

{

return p2;

}

void Line::setFirstPoint(Point p)

{

p1 = p;

}

void Line::setSecondPoint(Point p)

{

p2 = p;

}

void Line::draw(Graphics^ g)

{

g->DrawLine(gcnew Pen(Color::Black, 7), p1, p2);

}

Shape^ Line::find(int x, int y)

{

if (x < fmin(p1.X, p2.X) - 3 || x > fmax(p1.X, p2.X) + 3 || y < fmin(p1.Y, p2.Y) - 3 || y > fmax(p1.Y, p2.Y) + 3)

return nullptr;

int ab = abs((p1.Y - p2.Y) \* x + (p2.X - p1.X) \* y + (p1.X \* p2.Y - p2.X \* p1.Y));

if (sqrt(abs((p1.Y - p2.Y) \* x + (p2.X - p1.X) \* y + (p1.X \* p2.Y - p2.X \* p1.Y))) <= 49)

return this;

return nullptr;

}

Содержание файла Circle.h:

#pragma once

#include "Shape.h"

ref class Circle :

public Shape

{

System::Drawing::Rectangle rect;

public:

Circle(System::Drawing::Rectangle rect);

System::Drawing::Rectangle getRect();

void setRect(System::Drawing::Rectangle rect);

// Унаследовано через Shape

virtual void draw(System::Drawing::Graphics^ g) override;

// Унаследовано через Shape

virtual Shape^ find(int x, int y) override;

};

Содержание файла Circle.cpp:

#include "Circle.h"

#include <cmath>

Circle::Circle(Rectangle rect)

{

this->rect = rect;

type = 3;

}

Rectangle Circle::getRect()

{

return rect;

}

void Circle::setRect(Rectangle rect)

{

this->rect = rect;

}

void Circle::draw(System::Drawing::Graphics^ g)

{

g->DrawEllipse(gcnew Pen(Color::Black, 5), rect);

}

Shape^ Circle::find(int x, int y)

{

int centerX = (rect.Left + rect.Right) / 2;

int centerY = (rect.Top + rect.Bottom) / 2;

int a = rect.Width / 2;

int b = rect.Height / 2;

double f = pow((double)(x - centerX) / a, 2) + pow((double)(y - centerY) / b, 2);

if (f >= 0.8 && f <= 1.17)

return this;

return nullptr;

}

Содержание файла DrawingField.h:

#pragma once

#include <cliext/list>

#include "Shape.h"

using namespace cliext;

ref class DrawingField

{

private:

list<Shape^> shapes;

public:

void addShape(Shape^ shape);

void drawShapes(Graphics^ g);

Shape^ findShape(int x, int y);

list<Shape^> getShapes();

void setShapes(list<Shape^> shapes);

void clearList();

void deleteShape(Shape^ shape);

};

Содержание файла DrawingField.cpp:

#include "DrawingField.h"

void DrawingField::addShape(Shape^ shape)

{

shapes.push\_front(shape);

}

void DrawingField::drawShapes(Graphics^ g)

{

if (!shapes.empty()) {

for each (Shape ^ shape in shapes) {

shape->draw(g);

}

}

}

Shape^ DrawingField::findShape(int x, int y)

{

Shape^ result = nullptr;

for each (Shape^ shape in shapes)

if (shape->find(x, y) != nullptr)

{

result = shape;

shapes.remove(shape);

break;

}

return result;

}

list<Shape^> DrawingField::getShapes()

{

return shapes;

}

void DrawingField::clearList()

{

shapes.clear();

}

void DrawingField::setShapes(list<Shape^> shapes)

{

this->shapes = shapes;

}

void DrawingField::deleteShape(Shape^ shape)

{

shapes.remove(shape);

}

Содержание файла Serializer.h:

#pragma once

#include "Line.h"

#include "Circle.h"

#include "Rect.h"

#include <pugi/pugixml.hpp>

#include "DrawingField.h"

#define NAME\_OF(var) #var

ref class Serializer

{

public:

void serialize(pugi::xml\_node parent, std::string const& name, int value);

void serialize(pugi::xml\_node parent, std::string const& name, DrawingField^ field);

void serialize(pugi::xml\_node parent, std::string const& name, Line^ line);

void serialize(pugi::xml\_node parent, std::string const& name, Circle^ circle);

void serialize(pugi::xml\_node parent, std::string const& name, Rect^ rect);

void serialize(pugi::xml\_node parent, std::string const& name, int index, Point point);

void serialize(pugi::xml\_node parent, std::string const& name, Point point);

void serialize(pugi::xml\_node parent, std::string const& name, System::Drawing::Size size);

private:

template<typename T>

T const& toXml(T const& v);

static char const\* toXml(std::string const& v);

static pugi::xml\_node namedChild(pugi::xml\_node parent, std::string const& name);

};

Содержание файла Serializer.cpp:

#include "Serializer.h"

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, int value)

{

pugi::xml\_node n = namedChild(parent, name);

n.text().set(value);

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, DrawingField^ field)

{

pugi::xml\_node f = namedChild(parent, name);

list<Shape^> shapes = field->getShapes();

for each (Shape^ shape in shapes)

{

switch(shape->type)

{

case 1:

{

Line^ line = (Line^)shape;

serialize(f, NAME\_OF(Line), line);

break;

}

case 2:

{

Rect^ rect = (Rect^)shape;

serialize(f, NAME\_OF(Rect), rect);

break;

}

case 3:

{

Circle^ circle = (Circle^)shape;

serialize(f, NAME\_OF(Circle), circle);

break;

}

}

}

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, Line^ line)

{

pugi::xml\_node l = namedChild(parent, name);

serialize(l, NAME\_OF(Type), line->type);

serialize(l, NAME\_OF(Point), 1, line->getFirstPoint());

serialize(l, NAME\_OF(Point), 2, line->getSecondPoint());

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, Circle^ circle)

{

pugi::xml\_node c = namedChild(parent, name);

Point point(circle->getRect().X, circle->getRect().Y);

System::Drawing::Size size(circle->getRect().Width, circle->getRect().Height);

pugi::xml\_node Rectangle = namedChild(c, NAME\_OF(Rectangle));

serialize(c, NAME\_OF(Type), circle->type);

serialize(Rectangle, NAME\_OF(Point), point);

serialize(Rectangle, NAME\_OF(Size), size);

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, Rect^ rect)

{

pugi::xml\_node r = namedChild(parent, name);

Point point(rect->getRect().X, rect->getRect().Y);

System::Drawing::Size size(rect->getRect().Width, rect->getRect().Height);

pugi::xml\_node Rectangle = namedChild(r, NAME\_OF(Rectangle));

serialize(r, NAME\_OF(Type), rect->type);

serialize(Rectangle, NAME\_OF(Point), point);

serialize(Rectangle, NAME\_OF(Size), size);

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, int index, Point point)

{

pugi::xml\_node p = namedChild(parent, name);

p.append\_attribute("index");

p.attributes\_begin()->set\_value(index);

serialize(p, "X", point.X);

serialize(p, "Y", point.Y);

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, Point point)

{

pugi::xml\_node p = namedChild(parent, name);

serialize(p, "X", point.X);

serialize(p, "Y", point.Y);

}

void Serializer::serialize(pugi::xml\_node parent, std::string const& name, System::Drawing::Size size)

{

pugi::xml\_node s = namedChild(parent, name);

serialize(s, "Width", size.Width);

serialize(s, "Height", size.Height);

}

template<typename T>

T const& Serializer::toXml(T const& v) {

return v;

}

char const\* Serializer::toXml(std::string const& v) {

return v.c\_str();

}

pugi::xml\_node Serializer::namedChild(pugi::xml\_node parent, std::string const& name) {

pugi::xml\_node child = parent.append\_child();

child.set\_name(name.c\_str());

return child;

}

Содержание файла Deserializer.h:

#pragma once

#include <pugi/pugixml.hpp>

#include "DrawingField.h"

#include "Circle.h"

#include "Line.h"

#include "Rect.h"

#define NAME\_OF(var) #var

ref class Deserializer

{

public:

void deserialize(pugi::xml\_node parent, std::string const& name, DrawingField %field);

};

Содержание файла Deserializer.cpp:

#include "Deserializer.h"

void Deserializer::deserialize(pugi::xml\_node parent, std::string const& name, DrawingField %field)

{

pugi::xml\_node f = parent.first\_element\_by\_path(name.c\_str());

list<Shape^> shapes;

for (const auto& child : parent.child(NAME\_OF(DrawingField)))

{

int type = child.first\_element\_by\_path(NAME\_OF(Type)).text().as\_int();

switch (type)

{

case 1:

{

Point p1;

Point p2;

pugi::xml\_node point = child.first\_element\_by\_path(NAME\_OF(Point));

p1.X = point.first\_element\_by\_path(NAME\_OF(X)).text().as\_int();

p1.Y = point.first\_element\_by\_path(NAME\_OF(Y)).text().as\_int();

point = point.next\_sibling();

p2.X = point.first\_element\_by\_path(NAME\_OF(X)).text().as\_int();

p2.Y = point.first\_element\_by\_path(NAME\_OF(Y)).text().as\_int();

shapes.push\_back(gcnew Line(p1, p2));

break;

}

case 2:

{

pugi::xml\_node rectangle = child.first\_element\_by\_path(NAME\_OF(Rectangle));

pugi::xml\_node point = rectangle.first\_element\_by\_path(NAME\_OF(Point));

Point p;

p.X = point.first\_element\_by\_path(NAME\_OF(X)).text().as\_int();

p.Y = point.first\_element\_by\_path(NAME\_OF(Y)).text().as\_int();

pugi::xml\_node size = rectangle.first\_element\_by\_path(NAME\_OF(Size));

Size s;

s.Width = size.first\_element\_by\_path(NAME\_OF(Width)).text().as\_int();

s.Height = size.first\_element\_by\_path(NAME\_OF(Height)).text().as\_int();

shapes.push\_back(gcnew Rect(System::Drawing::Rectangle(p, s)));

break;

}

case 3:

{

pugi::xml\_node rectangle = child.first\_element\_by\_path(NAME\_OF(Rectangle));

pugi::xml\_node point = rectangle.first\_element\_by\_path(NAME\_OF(Point));

Point p;

p.X = point.first\_element\_by\_path(NAME\_OF(X)).text().as\_int();

p.Y = point.first\_element\_by\_path(NAME\_OF(Y)).text().as\_int();

pugi::xml\_node size = rectangle.first\_element\_by\_path(NAME\_OF(Size));

Size s;

s.Width = size.first\_element\_by\_path(NAME\_OF(Width)).text().as\_int();

s.Height = size.first\_element\_by\_path(NAME\_OF(Height)).text().as\_int();

shapes.push\_back(gcnew Circle(System::Drawing::Rectangle(p, s)));

break;

}

}

}

field.setShapes(shapes);

}