**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Отчет по индивидуальному домашнему заданию

по ОБЪЕКТО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

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

Студент \_\_\_\_\_\_\_\_\_\_\_ Сурова Д.А.

Группа ПМ-21-1

Руководитель

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

Липецк 2022

# **Оглавление:**

Цель работы…………………………………………………………………....3

Задание кафедры………………………………………………………………4

Код………………..…………………………………………………………….5

Пример работы программы…………………………………………………..10

Ответы на контрольные вопросы……………………………………………12

Вывод………………………………………………………………………….17

# **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 | Редактор |
| 14 | Планировка магазина |

# **Код:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Планировка\_магазина1.\_2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent(); //инициалиация компонентов

}

PictureBox pct; //объявление PictureBox

public int sch = 0; //заводим счетчик для того, чтобы была возможность создавать несколько элементов

private void button1\_Click(object sender, EventArgs e) //обработка кнопки кассы

{

pct = new PictureBox();

pct.Height = 119; // параметры элемента

pct.Width = 128;

pct.Name = "pct" + sch;

pct.Left = 20 + sch \* 110; //координаты появления элемента

pct.Top = 30;

pct.Image = Properties.Resources.kassa; //прикрепляем картинку к PictureBox

panel1.Controls.Add(pct); //добавляем элемент на панель

sch += 1;

pct.MouseUp += pct\_MouseUp; //ссылки на события

pct.MouseDown += pct\_MouseDown;

pct.MouseMove += pct\_MouseMove;

}

//реализация движения объектов

private bool Dragging1;

private int xPos1; //координаты элемента

private int yPos1;

private void pct\_MouseUp(object sender, MouseEventArgs e) //если элемент не выбран

{

Dragging1 = false;

}

private void pct\_MouseDown(object sender, MouseEventArgs e) //добавление элемента

{

if (e.Button == MouseButtons.Left) //если нажатa левая кнока мыши

{

Dragging1 = true;

xPos1 = e.X;

yPos1 = e.Y;

}

}

private void pct\_MouseMove(object sender, MouseEventArgs e) // перенесение элемента по схеме

{

Control c = sender as Control;

if (Dragging1 && c != null)

{

c.Top = e.Y + c.Top - yPos1;

c.Left = e.X + c.Left - xPos1;

}

}

PictureBox pictureBox1; //объявление PictureBox1

public int sch1 = 0;//заводим счетчик для того, чтобы была возможность создавать несколько элементов

private void button3\_Click(object sender, EventArgs e) //обработка кнопки холодильника

{

pictureBox1 = new PictureBox();

pictureBox1.Height = 200; //параметры элемента

pictureBox1.Width = 85;

pictureBox1.Name = "pictureBox1" + sch1;

pictureBox1.Left = 20 + sch1 \* 110; //координаты появления элемента

pictureBox1.Top = 30;

pictureBox1.Image = Properties.Resources.holod;//прикрепляем картинку к PictureBox1

panel1.Controls.Add(pictureBox1); //добавляем элемент на панель

sch1 += 1;

pictureBox1.MouseUp += pictureBox1\_MouseUp\_1; //ссылки на события

pictureBox1.MouseDown += pictureBox1\_MouseDown\_1;

pictureBox1.MouseMove += pictureBox1\_MouseMove\_1;

}

//реализация движения элемента

private bool Dragging;

private int xPos;

private int yPos;

private void pictureBox1\_MouseUp\_1(object sender, MouseEventArgs e) //если элемент не выбран

{

Dragging = false;

}

private void pictureBox1\_MouseDown\_1(object sender, MouseEventArgs e) //добавление элемента

{

if (e.Button == MouseButtons.Left) //если мы нажимием на левую кнопку мыши

{

Dragging = true;

xPos = e.X;

yPos = e.Y;

}

}

private void pictureBox1\_MouseMove\_1(object sender, MouseEventArgs e) //перемещение элемента

{

Control c = sender as Control;

if (Dragging && c != null)

{

c.Top = e.Y + c.Top - yPos;

c.Left = e.X + c.Left - xPos;

}

}

PictureBox pictureBox2; //объявление PictureBox2

public int sch2 = 0;//заводим счетчик для того, чтобы была возможность создавать несколько элементов

private void button4\_Click(object sender, EventArgs e) //обработка кнопки 4

{

pictureBox2 = new PictureBox();

pictureBox2.Height = 120; //параметры элемента

pictureBox2.Width = 120;

pictureBox2.Name = "pictureBox2" + sch2;

pictureBox2.Left = 20 + sch2 \* 110; //координаты появления элемента

pictureBox2.Top = 30;

pictureBox2.Image = Properties.Resources.showcase1; //прикрепляем картинку к PictureBox2

panel1.Controls.Add(pictureBox2);//добавляем элемент на панель

sch2 += 1;

pictureBox2.MouseUp += pictureBox2\_MouseUp\_2; //ссылки на события

pictureBox2.MouseDown += pictureBox2\_MouseDown\_2;

pictureBox2.MouseMove += pictureBox2\_MouseMove\_2;

}

//реализация движения элемента

private bool Dragging2;

private int xPos2;

private int yPos2;

private void pictureBox2\_MouseUp\_2(object sender, MouseEventArgs e)//если элемент не выбран

{

Dragging2 = false;

}

private void pictureBox2\_MouseDown\_2(object sender, MouseEventArgs e)//добавление элемента

{

if (e.Button == MouseButtons.Left)

{

Dragging2 = true;

xPos2 = e.X;

yPos2 = e.Y;

}

}

private void pictureBox2\_MouseMove\_2(object sender, MouseEventArgs e)//перемещение элемента

{

Control c = sender as Control;

if (Dragging2 && c != null)

{

c.Top = e.Y + c.Top - yPos2;

c.Left = e.X + c.Left - xPos2;

}

}

PictureBox pictureBox3;//объявление PictureBox1

public int sch3 = 0;//заводим счетчик для того, чтобы была возможность создавать несколько элементов

private void button2\_Click(object sender, EventArgs e)//обработка кнопки 2

{

pictureBox3 = new PictureBox();

pictureBox3.Height = 120;//параметры элемента

pictureBox3.Width = 120;

pictureBox3.Name = "pictureBox3" + sch3;

pictureBox3.Left = 20 + sch3 \* 110; //координаты появления элемента

pictureBox3.Top = 30;

pictureBox3.Image = Properties.Resources.showcase3; //прикрепляем картинку к PictureBox

panel1.Controls.Add(pictureBox3); //добавление элемента на панель

sch3 += 1;

pictureBox3.MouseUp += pictureBox3\_MouseUp\_3; //ссылки на события

pictureBox3.MouseDown += pictureBox3\_MouseDown\_3;

pictureBox3.MouseMove += pictureBox3\_MouseMove\_3;

}

//реализация перемещения элемента

private bool Dragging3;

private int xPos3;

private int yPos3;

private void pictureBox3\_MouseUp\_3(object sender, MouseEventArgs e) //если элемент не выбран

{

Dragging3 = false;

}

private void pictureBox3\_MouseDown\_3(object sender, MouseEventArgs e) //добавление элемента

{

if (e.Button == MouseButtons.Left)

{

Dragging3 = true;

xPos3 = e.X;

yPos3 = e.Y;

}

}

private void pictureBox3\_MouseMove\_3(object sender, MouseEventArgs e)//перемещение элемента

{

Control c = sender as Control;

if (Dragging3 && c != null)

{

c.Top = e.Y + c.Top - yPos3;

c.Left = e.X + c.Left - xPos3;

}

}

//реализация кнопок удаления элементов

private void button5\_Click(object sender, EventArgs e)

{

pct.Dispose();

}

private void button6\_Click(object sender, EventArgs e)

{

pictureBox1.Dispose();

sch1 = 0;

}

private void button7\_Click(object sender, EventArgs e)

{

pictureBox2.Dispose();

sch2 = 0;

}

private void button8\_Click(object sender, EventArgs e)

{

pictureBox3.Dispose();

sch3 = 0;

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

**Пример выполнения программы**

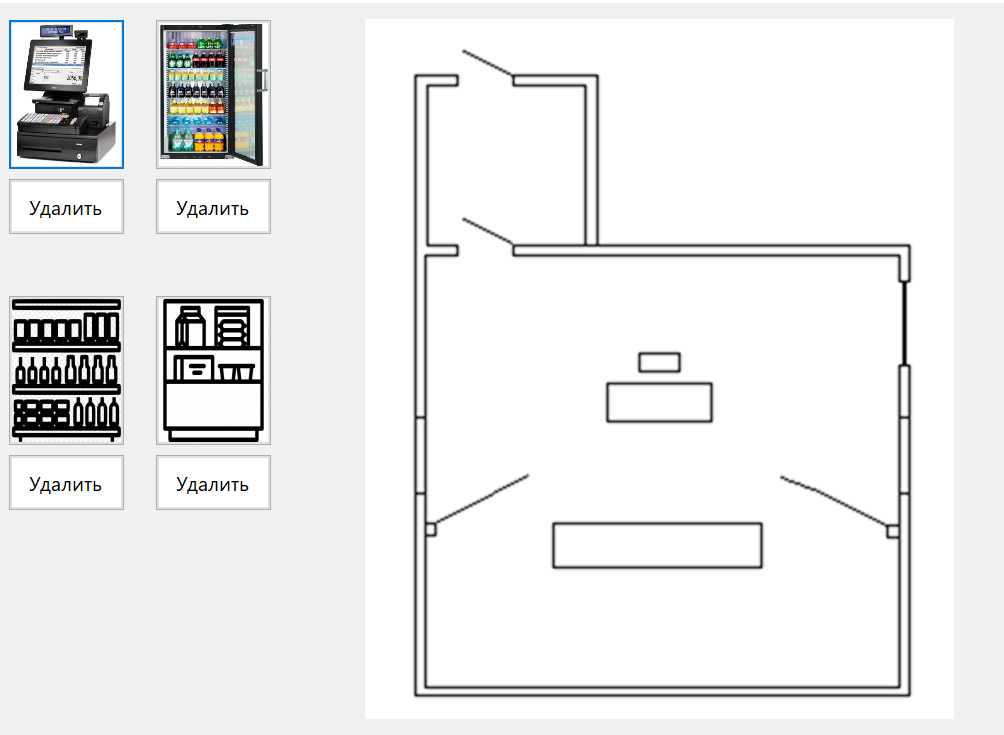


Рисунок 1. Запуск программы

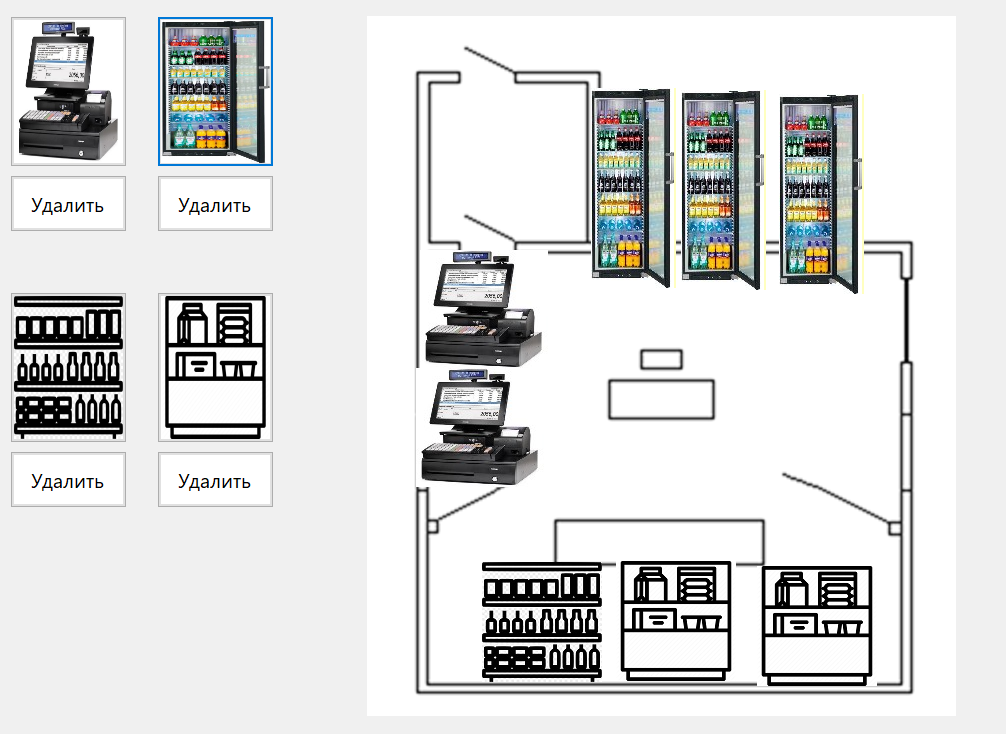


Рисунок 2. Пример заполнения схемы

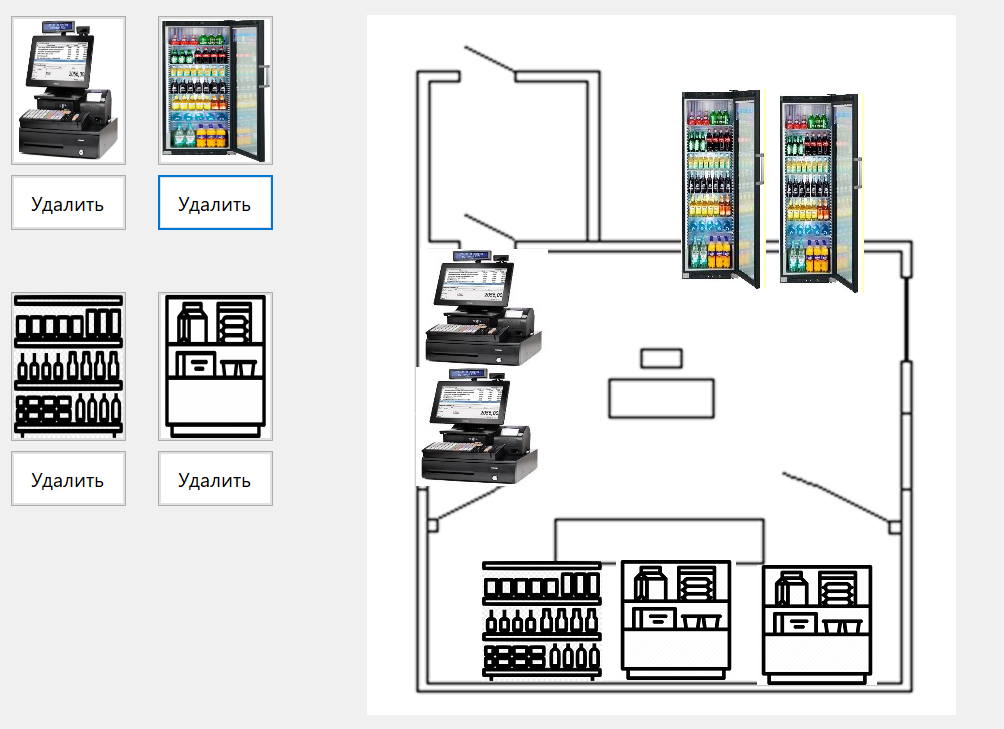


Рисунок 3. Удаление холодильника

# **Ответы на контрольные вопросы:**

# **Зачем нужен перегруженный оператор присваивания?**

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

# **Зачем нужен механизм наследования?**

# Наследование — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены. 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.

# **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.

|  |  |  |  |
| --- | --- | --- | --- |
| Модификатор наследования | Модификатор доступа | | |
| public | private | protected |
| public-наследование | public | private | protected |
| private-наследование | private | private | private |
| protected-наследование | protected | private | protected |

# **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, применительно к объектно-ориентированному программированию, 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. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор. Вам, как программисту, не нужно делать этот выбор самому. Нужно только помнить и использовать общий интерфейс.

# Полиморфизм может применяться также и к операторам. Фактически во всех языках программирования ограниченно применяется полиморфизм, например, в арифметических операторах. В С++ вы можете применить эту концепцию к заданным вами типам данных. Такой тип полиморфизма называется перегрузкой операторов.

# **Что понимается под динамическим полиморфизмом?**

# 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.

# **Что такое интерфейс класса?**

# Интерфейс — это класс, который не имеет переменных-членов и все методы которого являются чистыми виртуальными функциями! Интерфейсы еще называют «классами-интерфейсами» или «интерфейсными классами».

# Интерфейсные классы принято называть с I в начале.

# **Зачем нужен чисто виртуальный метод?**

# Чисто виртуальная функция является функцией, которая объявляется в базовом классе, но не имеет в нем определения. Поскольку она 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.

# **Какой класс называется абстрактным?**

# Если какой-либо класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется абстрактным. Важной особенностью абстрактных классов является то, что не существует ни одного объекта данного класса. Вместо этого абстрактный класс служит в качестве базового для других производных классов. Причина, по которой абстрактный класс не может быть ис­пользован для объявления объекта, заключается в том, что одна или несколько его функций-членов не имеют определения. Тем не менее, даже если базовый класс является абстрактным, все равно можно объявлять указатели или ссылки на него, с помощью которых затем поддерживает­ся полиморфизм времени исполнения.

# **Вывод:**

В данной работе я закрепила навыки использования механизмов ООП на примере реализации графического редактора.