|  |  |
| --- | --- |
|  | **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**  **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Институт (факультет) | Факультет автоматизации и информатики |
| Кафедра | Автоматизированные системы управления |

Отчет по индивидуальному домашнему заданию

По дисциплине: «Объектно-Ориентированное Программирование

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ПМ-22-1 |  |  |  | Ноздреватых И.С |
|  | группа |  | подпись, дата |  | фамилия, инициалы |
|  |  |  |  |  |  |
| Руководитель |  |  |  |  |  |
| к.т.н., доцент | |  |  |  | Кургасов В.В |
| ученая степень, ученое звание | |  | подпись, дата |  | фамилия, инициалы |

Липецк 2023

Оглавление

[Цель работы: 3](#_Toc58691169)

[Задание кафедры: 3](#_Toc58691170)

[Вариант № 7: 3](#_Toc58691171)

[Исходный код: 4](#_Toc58691172)

[Результат работы. 10](#_Toc58691173)

[Контрольные вопросы: 12](#_Toc58691174)

[Вывод: 16](#_Toc58691175)

# **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++.

# **Задание варианта:**

|  |  |
| --- | --- |
| Вариант | Редактор |
| 7 | Планировка кухни |

# Исходный код:

MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace WpfApp3

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

//Инициализируем переменные класса

int numb = 0; //Элемент массива мебели

Vector relativeMousePos; // Координаты мыши

FrameworkElement draggedObject; //Элемент движения(наша мебель), которая апкастнута до класса FrameworkElement

List<Border> Mebel = new List<Border>(); //Массив с нашей мебелью

public MainWindow()

{

InitializeComponent();

}

//Функция захвата изображения

void StartDrag(object sender, MouseButtonEventArgs e)

{

draggedObject = (FrameworkElement)sender;//Приравниваем к элементу движения инициализированному выше объект, который подписан на эту функцию и отправил

//запрос на начало функции

relativeMousePos = e.GetPosition(draggedObject) - new Point();//Задаём ккординаты мыши

draggedObject.MouseMove += OnDragMove; //Подписываемся на функцию движения объекта мышкой

draggedObject.LostMouseCapture += OnLostCapture;//Подписываемся на сброс фокуса

draggedObject.MouseUp += OnMouseUp; //Подписываеся на событие поднятия кнопки мыши вверх

Mouse.Capture(draggedObject);//Запускаем функцию Capture на нашей мышке, передавая туда элемент движения(для работы с элементом, на который мы кликнули)

}

//Функция движения

void OnDragMove(object sender, MouseEventArgs e)

{

UpdatePosition(e); //Вызывает функцию обновления позиции, в зависимости от координат мыши.

}

//Функции обновления позиции

void UpdatePosition(MouseEventArgs e)

{

var point = e.GetPosition(mainField);//Получаем координаты мыши отностилеьно всей формы

var newPos = point - relativeMousePos;//Задаём новые координаты(координаты мыши относительно Canvas

Canvas.SetLeft(draggedObject, newPos.X);//Передаём горизонтальные координаты в Canvas

Canvas.SetTop(draggedObject, newPos.Y);//Передаём вертикальные координаты в Canvas

}

//Функция поднятия кнопки мыши

void OnMouseUp(object sender, MouseButtonEventArgs e)

{

FinishDrag(sender, e);//Вызываем функцию окончания переноса

Mouse.Capture(null);//Запускаем функцию Capture на нашей мышке, передавая туда null, чтобы сбросить объект, с которым мы работаем

}

//Функция сброса фокуса

void OnLostCapture(object sender, MouseEventArgs e)

{

FinishDrag(sender, e);//Вызываем функцию прекращения захвата изображения, передавая туда наш элемент по подписке и свойства мыши (e)

}

//Функция прекращения захвата изображения

void FinishDrag(object sender, MouseEventArgs e)

{

//Отписываемся от всех функций, на которые подписывались в StartDrag

draggedObject.MouseMove -= OnDragMove;

draggedObject.LostMouseCapture -= OnLostCapture;

draggedObject.MouseUp -= OnMouseUp;

UpdatePosition(e);//Последний раз обновляем позицию картинки перед сбросом

}

//Функция фокуса на элементе для рисования рамки вокруг выбранного элемента

private void getFocusOnObject(object sender, MouseButtonEventArgs e)

{

LostBorder(Mebel[numb]);//Вызываем функцию сброса рамки, передавая туда элемент

Border figura = sender as Border;//Получаем сам элемент

numb = int.Parse((string)figura.Tag);//Получаем тэг этого элемента и превращаем в int, чтобы обраттиься к списку всех элементов по индексу

DrawBorder(Mebel[numb]);//Рисуем рамку вокруг выбранного элемента

}

//Функция удаления объекта оп нажатию правой кнопки мыши

private void killObject(object sender, MouseButtonEventArgs e)

{

Border figura = sender as Border; //Получаем сам элемент

numb = int.Parse((string)figura.Tag);//Получаем тэг этого элемента и превращаем в int, чтобы обраттиься к списку всех элементов по индексу

Mebel.Remove(Mebel[numb]);//Удаляем из списка всех элементов

mainField.Children.Remove(figura);//Удаляем с Canvas

}

private void Add(object sender, RoutedEventArgs e)//добавление

{

string str = "";//Инициализируем строку пути к элементу

//Проверяем какой элемент из списка выбран и присваемваем путь к картинке

if ((bool)RefrigeratorRB.IsChecked) str = "pics/1.png";

if ((bool)FireRB.IsChecked) str = "pics/2.png";

if ((bool)WaterRB.IsChecked) str = "pics/3.png";

if ((bool)BoxRB.IsChecked) str = "pics/4.png";

if ((bool)TableRB.IsChecked) str = "pics/5.png";

AddPattern(str);//Вызываем функци добавления по паттерну

}

//Функция рисования рамок

private void DrawBorder(object sender)

{

var b = (Border)sender;//получаем элемент

b.BorderThickness = new Thickness(1);//Задаём ему свойство толщины границ равным 1 пикселю

}

//Функция убирания рамок

private void LostBorder(object sender)

{

var b = (Border)sender;//получаем элемент

b.BorderThickness = new Thickness(0);//Задаём ему свойство толщины границ равным 0 пикселей, т.е. убираем

}

//Функция добавления на Canvas по паттерну

private void AddPattern(string pattern)

{

var myBrush = new ImageBrush();//Инициализируем кисть для рисования

var border = new Border//Инициализируем границу

{

BorderBrush = Brushes.Aqua,//задаём границе цвет

BorderThickness = new Thickness(0)//и толщину. Изначально равную 0, т.к. только добавленный элемент не выбран

};

var SomeImg = new Image//Инициализируем картинку

{

Source = new BitmapImage(new Uri(@pattern, UriKind.Relative)),//передаём путь полученный в функци Add, чтобы нарисовать картинку

Width = 100,//Задаём ширину

Height = 100,//И высоту

};

border.Child = SomeImg;//Добавляем картинку в дочерние элементы к границе

Mebel.Add(border);//Добавляем картинку в массив картинок

border.Tag = (Mebel.Count - 1).ToString();//Задаём ей тэг с равным её индексу

myBrush.ImageSource = SomeImg.Source; //Рисуем картинку кистью инициализированной выше

border.MouseLeftButtonDown += new MouseButtonEventHandler(getFocusOnObject); //Подписываем на функцию рисования рамки

border.MouseLeftButtonDown += new MouseButtonEventHandler(StartDrag);//Подписываем на фукнцию начала захвата

border.MouseRightButtonDown += new MouseButtonEventHandler(killObject);//Подписываем на функцию удаления объекта

Canvas.SetLeft(border, 100);//Задаём нашей картинке координаты на Canvas по горизонтали

Canvas.SetTop(border, 100);//и по вертикали

mainField.Children.Add(border);//И добавляем на сам Canvas

}

//Функция перемещения слайдера изменения градуса поворота

private void Slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

if(Mebel.Count != 0)//Проверяем есть ли у нас хоть один элемент

{

//Если есть

RotateTransform rotate = new RotateTransform(AngleSlider.Value);//Создаём объект поворота картинки и передаём туда цифру в градусах(взятую из слайдера)

rotate.CenterX = 50;//Указываем центр (50;50), т.к. наша картинка 100x100 и её центр находится в (50;50)

rotate.CenterY = 50;

AngleTextValue.Content = AngleSlider.Value.ToString()+ "°";//Изменяем текст картинки на значения градусов на слайдере

Mebel[numb].RenderTransform = rotate;//Поворачиваем нашу картинку

}

}

}

}

MainWindow.xaml

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp3" x:Class="WpfApp3.MainWindow"

mc:Ignorable="d"

Title="Планировка кухни" WindowStyle="SingleBorderWindow" Height="450" Width="810" MinWidth="800" MinHeight="450" MaxWidth="810" MaxHeight="450" Background="GhostWhite">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="60\*"/>

<ColumnDefinition Width="611\*"/>

<ColumnDefinition Width="139\*"/>

</Grid.ColumnDefinitions>

<Border BorderBrush="Black" BorderThickness="10" CornerRadius="10" Grid.ColumnSpan="3" Margin="0,0,139,0">

<Canvas x:Name="mainField" Background="#FFBBC6C8" Focusable ="True" MaxWidth="656" MaxHeight="419" ClipToBounds="True">

</Canvas>

</Border>

<Grid HorizontalAlignment="Left" Height="409" VerticalAlignment="Top" Width="121" Grid.Column="2" Margin="5,10,0,0">

<ListBox HorizontalAlignment="Left" Height="117" VerticalAlignment="Bottom" Width="111" Margin="10,0,0,282">

<RadioButton x:Name="RefrigeratorRB" Content="Холодильник" HorizontalAlignment="Left" VerticalAlignment="Top" Background="White" IsChecked="True"/>

<RadioButton x:Name="FireRB" Content="Плита" HorizontalAlignment="Left" VerticalAlignment="Top" Background="White"/>

<RadioButton x:Name="WaterRB" Content="Рукомойник" Width="101"/>

<RadioButton x:Name="TableRB" Content="Стол" Width="101"/>

<RadioButton x:Name="BoxRB" Content="Шкаф" Width="101"/>

</ListBox>

<Button Content="Добавить" Click="Add" Height="56" Canvas.Left="669" Canvas.Top="149" Margin="18,139,10,214"/>

<Slider x:Name="AngleSlider" HorizontalAlignment="Center" Margin="0,215,0,0" VerticalAlignment="Top" Width="101" Maximum="360" ValueChanged="Slider\_ValueChanged"/>

<Label Content="0°" Margin="10,233,90,0" VerticalAlignment="Top" RenderTransformOrigin="0.623,0.31"/>

<Label Content="360°" HorizontalAlignment="Left" Margin="85,233,0,0" VerticalAlignment="Top"/>

<Label Content="Угол:" HorizontalAlignment="Left" Margin="37,284,0,0" VerticalAlignment="Top"/>

<Label Content="0°" x:Name="AngleTextValue" Margin="68,283,19,0" VerticalAlignment="Top"/>

</Grid>

</Grid>

</Window>

# **Результат работы**

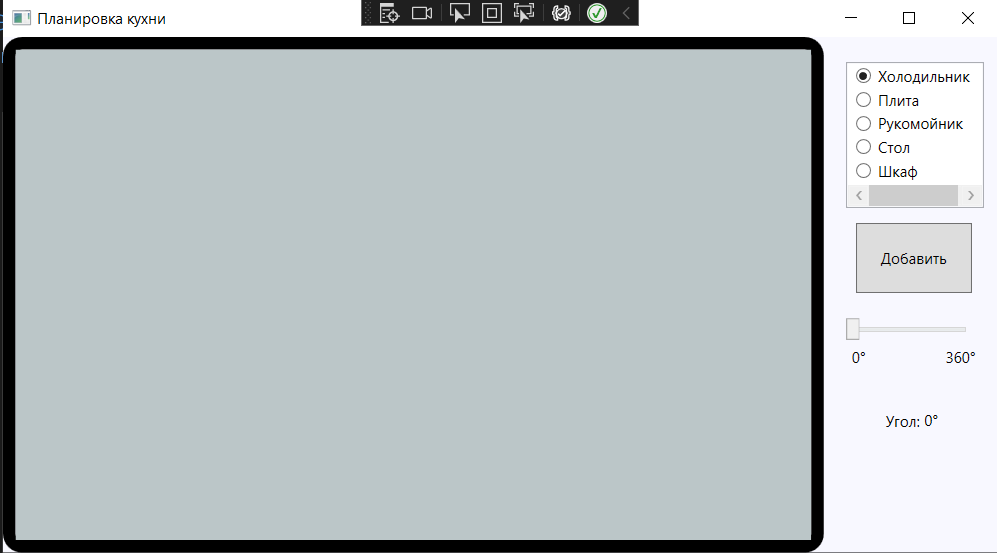


Рисунок 1. Запуск программы.

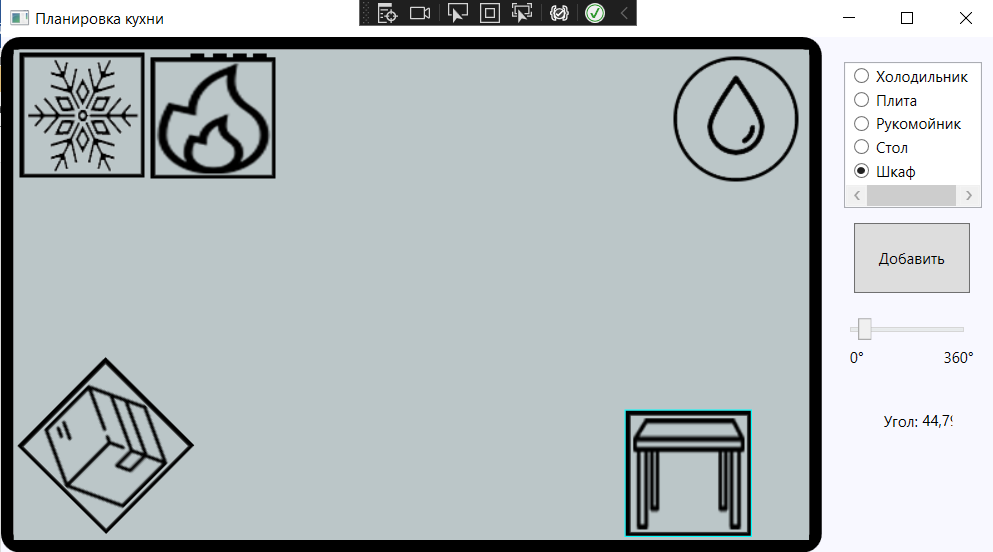


Рисунок 2. Чтобы получить данную расстановку, необходимо перетащить предметы с поля «мебель» на планировку.

# Контрольные вопросы:

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

Оператор присваивания = - может быть перегружен для конкретного класса.

Общая форма оператора присваивания для класса с именем CMyClass имеет вид: 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. В объектно-ориентированном программировании, начиная с Simula 67, абстрактные типы данных называются классами.

Суперкласс (англ. super class), родительский класс (англ. parent class), предок, родитель или надкласс — класс, производящий наследование в подклассах, т. е. класс, от которого наследуются другие классы. Суперклассом может быть подкласс, базовый класс, абстрактный класс и интерфейс.

Подкласс (англ. subclass), производный класс (англ. derived class), дочерний класс (англ. child class), класс потомок, класс наследник или класс-реализатор — класс, наследуемый от суперкласса или интерфейса, т. е. класс определённый через наследование от другого класса или нескольких таких классов. Подклассом может быть суперкласс.

Базовый класс (англ. base class) — это класс, находящийся на вершине иерархии наследования классов и в основании дерева подклассов, т. е. не являющийся подклассом и не имеющий наследований от других суперклассов или интерфейсов. Базовым классом может быть абстрактный класс и интерфейс. Любой не базовый класс является подклассом.

Интерфейс (англ. interface) — это структура, определяющая чистый интерфейс класса, состоящий из абстрактных методов. Интерфейсы участвуют в иерархии наследований классов и интерфейсов.

Суперинтерфейс (англ. super interface) или интерфейс-предок — это аналог суперкласса в иерархии наследований, т. е. это интерфейс производящий наследование в подклассах и подинтерфейсах.

Интерфейс-потомок, интерфейс-наследник или производный интерфейс (англ. derived interface) — это аналог подкласса в иерархии наследований интерфейсов, т. е. это интерфейс наследуемый от одного или нескольких суперинтерфейсов.

Базовый интерфейс — это аналог базового класса в иерархии наследований интерфейсов, т. е. это интерфейс, находящийся на вершине иерархии наследования.

Иерархия наследования или иерархия классов — дерево, элементами которого являются классы и интерфейсы.

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 — одна из трех основных парадигм ООП. Если говорить кратко, полиморфизм — это способность объекта использовать методы производного класса, который не существует на момент создания базового. 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.

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 абстрактный класс, у которого ни один метод не реализован, все они публичные и нет переменных класса.

Интерфейс нужен обычно, когда описывается только интерфейс (тавтология). Например, один класс хочет дать другому возможность доступа к некоторым своим методам, но не хочет себя «раскрывать». Поэтому он просто реализует интерфейс. Интерфейсные классы принято называть с I в начале.

1. **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.

1. **Какой класс называется абстрактным?**

Абстрактный класс используют, когда необходимо создать семейство классов (много разновидностей монстров в игре), при этом было бы лучше вынести общую реализацию и поведение в отдельный класс. При такой тактике переопределить/дописать придется только специфичные для каждого класса методы (у каждого монстра своя анимация удара/перемещения) и/или расширить функциональность класса.

Но возникает возможность создать абстрактный класс, что противоречит архитектуре: как "выделанная общая часть" может быть полноценным классом? Ответ: никак, надо запретить создавать подобный класс. Для этого указывают один из методов как чистый виртуальный ( pure virtual method ), пример:

virtual void f() = 0;

# **Вывод:**

В ходе выполнения данного индивидуального домашнего задания я закрепила навыки использования механизмов ООП на примере реализации графического редактора.