**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

По объектно-ориентированному программированию

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

Студент \_\_\_\_\_\_\_\_\_\_\_ Иванова Е.С.

Группа ПМ-21-1

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

**Вариант 6**

**Цель работы:** закрепить навыки использования механизмов ООП на примере реализации графического редактора.

**Задание кафедры:** реализовать на языке С++ редактор графической схемы (инфологическая схема). В ходе выполнения работы обязательно применение объектно-ориентированных возможностей языка C++:наследование 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++.

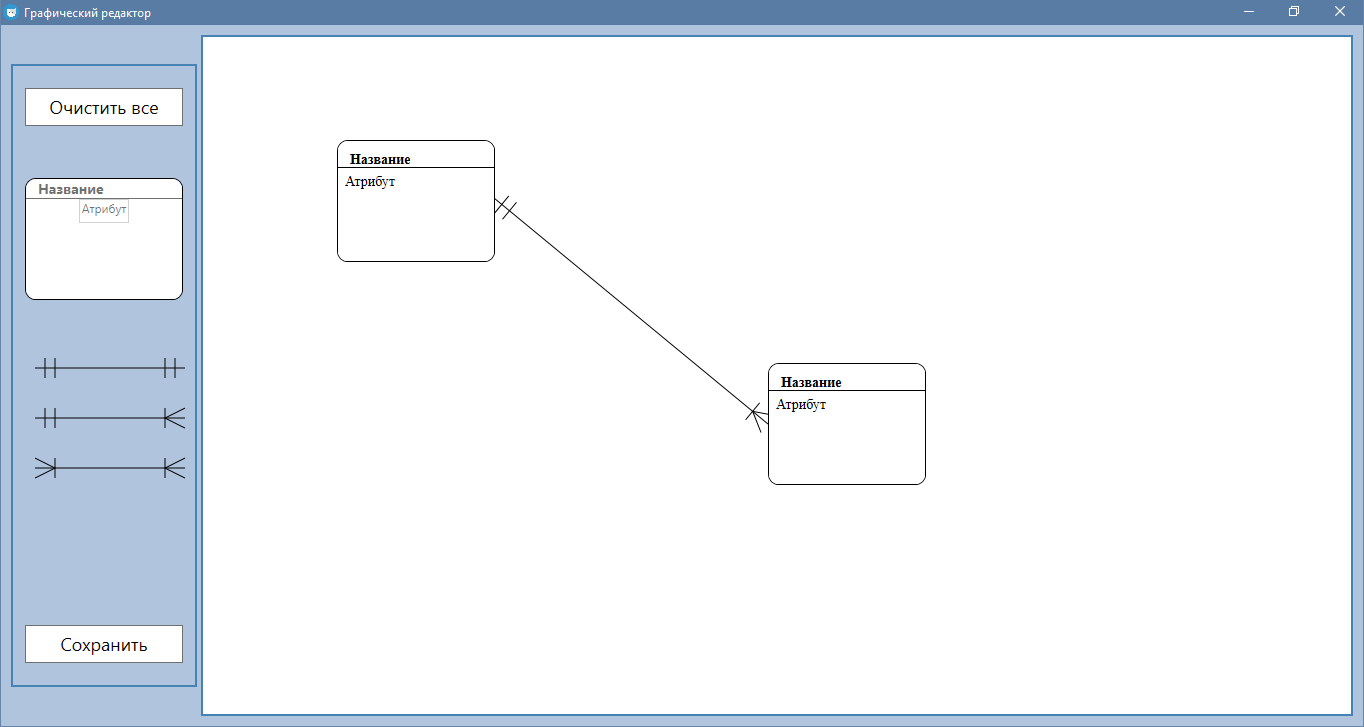
**Исходный код**

**Файл** [**MainWindow.xaml.cs**](https://github.com/EliavetaIvanova/oop333/blob/main/MainWindow.xaml.cs)

|  |
| --- |
| using System; |
|  | using System.Collections.Generic; |
|  | using System.IO; |
|  | 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 Editor |
|  | { |
|  | /// Главный класс приложения |
|  | public partial class MainWindow : System.Windows.Window |
|  | { |
|  | // префикс стрлки, который будет указываться при создании стрелки на поле |
|  | private string nameArrow = "arrow\_"; |
|  | // указывает за какую сторону в данный момент тянем стрелку |
|  | bool indexSideArrow; |
|  | // указывает индекс стрелки в массиве, которую в данный момент тянем |
|  | int indexCurrentArrow; |
|  | // указывает индекс окна в массиве, которое в данный момент передвигаем |
|  | int indexCurrentWindow; |
|  | // массив стрелок на поле |
|  | List<Arrow> myArrows; |
|  | // массив окон на поле |
|  | List<Window> myWindows; |
|  |  |
|  | // конструктор класса(инициализация переменных) |
|  | public MainWindow() |
|  | { |
|  | InitializeComponent(); |
|  | double radius = 20.0; |
|  | Config.setConfig(radius, borderdrawPanel.Margin.Top, borderdrawPanel.Margin.Left); |
|  |  |
|  | indexSideArrow = false; |
|  | indexCurrentArrow = -1; |
|  | indexCurrentWindow = -1; |
|  | myArrows = new List<Arrow>(); |
|  | myWindows = new List<Window>(); |
|  | } |
|  |  |
|  | // событие передвидения курсора по полю для рисования |
|  | private void drawPanel\_MouseMove(object sender, MouseEventArgs e) |
|  | { |
|  | // при нажатой кнопки |
|  | if (e.LeftButton == MouseButtonState.Pressed) |
|  | { |
|  | if (indexCurrentWindow != -1) |
|  | { |
|  | // передвигаем окно |
|  | myWindows[indexCurrentWindow].move(e.GetPosition(this)); |
|  | } |
|  | else if (indexCurrentArrow != -1) |
|  | { |
|  | // тянем стрелку за её начало |
|  | if (indexSideArrow) |
|  | { |
|  | // передвигаем стрелку |
|  | myArrows[indexCurrentArrow].setPoint1(e.GetPosition(this)); |
|  | } |
|  | else // тянем стрелку за её конец |
|  | { |
|  | // передвигаем стрелку |
|  | myArrows[indexCurrentArrow].setPoint2(e.GetPosition(this)); |
|  | } |
|  | // перерисовываем поля для рисования |
|  | drawPanel.InvalidateVisual(); |
|  | } |
|  | } |
|  | else // при отжатой кнопки |
|  | { |
|  | indexCurrentArrow = -1; |
|  | indexCurrentWindow = -1; |
|  |  |
|  | for (int count = 0; count < myWindows.Count; count++) |
|  | { |
|  | // проверяем навели ли курсором на окно |
|  | if (myWindows[count].equalWindowWithPoint(e.GetPosition(this))) |
|  | { |
|  | indexCurrentWindow = count; |
|  | myWindows[indexCurrentWindow].clearLinkArrows(); |
|  | foreach (Arrow myLine in myArrows) |
|  | { |
|  | myWindows[indexCurrentWindow].addLinkingArrowsInList(myLine); |
|  | } |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | if (indexCurrentWindow == -1) |
|  | { |
|  | int numberPointLineIndex = 0; |
|  | // проверяем навели ли курсором на один из концов стрелки |
|  | for (int count = 0; count < myArrows.Count; count++) |
|  | { |
|  | numberPointLineIndex = myArrows[count].equalLineWithPoint(e.GetPosition(this)); |
|  | if (numberPointLineIndex == 1) |
|  | { |
|  | indexCurrentArrow = count; |
|  | indexSideArrow = true; |
|  | break; |
|  | } |
|  | else if (numberPointLineIndex == 2) |
|  | { |
|  | indexCurrentArrow = count; |
|  | indexSideArrow = false; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Событие при котором отпустили кнопку мыши на поле для рисования |
|  | private void drawPanel\_MouseUp(object sender, MouseButtonEventArgs e) |
|  | { |
|  | if (indexCurrentArrow != -1) |
|  | { |
|  | // Если по близости с линией, которую только что перетаскивали есть |
|  | // другие линии, то соединяем их |
|  | foreach (Arrow line in myArrows) |
|  | { |
|  | if (!line.equalName(myArrows[indexCurrentArrow].name())) |
|  | { |
|  | Arrow.concateLines(myArrows[indexCurrentArrow], line); |
|  | } |
|  | } |
|  |  |
|  | int count = 0; |
|  | // Если по близости с линией, которую только что перетаскивали есть |
|  | // окна, то соединяем их |
|  | foreach (Window window in myWindows) |
|  | { |
|  | if (window.concateWindowWithLine(myArrows[indexCurrentArrow])) |
|  | { |
|  | ++count; |
|  | } |
|  |  |
|  | if (count >= 2) |
|  | { |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Событие DragAndDrop (когда отпускают кнопку мыши при перетаскивании объекта) |
|  | private void drawPanel\_Drop(object sender, DragEventArgs e) |
|  | { |
|  | string[] str = (string[])e.Data.GetFormats(); |
|  | // Если перетаскивали стрелку |
|  | if (str[0] == "System.Windows.Shapes.Line") |
|  | { |
|  | Line line = (Line)e.Data.GetData("System.Windows.Shapes.Line"); |
|  | Canvas canvas = (Canvas)line.Parent; |
|  |  |
|  | Line newLine = Arrow.copyLine(line, nameArrow + myArrows.Count, e.GetPosition(this)); |
|  | Line newEndDistantLine = Arrow.patternEndDistantLine(newLine); |
|  | Line newBeginDistantLine = Arrow.patternBeginDistantLine(newLine); |
|  |  |
|  | // Если стрелка первого типа, создаем её шаблон |
|  | if (line.Name == "Arrow\_type\_1") |
|  | { |
|  | Line newEndNearLine = Arrow.patternEndNearLine(newLine); |
|  | Line newBeginNearLine = Arrow.patternBeginNearLine(newLine); |
|  | myArrows.Add(new MyArrowType1(newLine, newEndDistantLine, newBeginDistantLine, newEndNearLine, newBeginNearLine)); |
|  | drawPanel.Children.Add(newEndNearLine); |
|  | drawPanel.Children.Add(newBeginNearLine); |
|  | Canvas.SetZIndex(newEndNearLine, 0); |
|  | Canvas.SetZIndex(newBeginNearLine, 0); |
|  | } |
|  | // Если стрелка второго типа, создаем её шаблон |
|  | else if (line.Name == "Arrow\_type\_2") |
|  | { |
|  | Polyline newPolyline = Arrow.patternEndPolyline(newLine); |
|  | Line newBeginNearLine = Arrow.patternBeginNearLine(newLine); |
|  | myArrows.Add(new Arrow2(newLine, newEndDistantLine, newBeginDistantLine, newBeginNearLine, newPolyline)); |
|  | drawPanel.Children.Add(newPolyline); |
|  | drawPanel.Children.Add(newBeginNearLine); |
|  | Canvas.SetZIndex(newPolyline, 0); |
|  | Canvas.SetZIndex(newBeginNearLine, 0); |
|  | } |
|  | // Если стрелка третьего типа, создаем её шаблон |
|  | else if (line.Name == "Arrow\_type\_3") |
|  | { |
|  | Polyline newEndPolyline = Arrow.patternEndPolyline(newLine); |
|  | Polyline newBeginPolyline = Arrow.patternBeginPolyline(newLine); |
|  | myArrows.Add(new Arrow3(newLine, newEndDistantLine, newBeginDistantLine, newEndPolyline, newBeginPolyline)); |
|  | drawPanel.Children.Add(newEndPolyline); |
|  | drawPanel.Children.Add(newBeginPolyline); |
|  | Canvas.SetZIndex(newEndPolyline, 0); |
|  | Canvas.SetZIndex(newBeginPolyline, 0); |
|  | } |
|  |  |
|  | drawPanel.Children.Add(newLine); |
|  | drawPanel.Children.Add(newEndDistantLine); |
|  | drawPanel.Children.Add(newBeginDistantLine); |
|  | Canvas.SetZIndex(newLine, 0); |
|  | Canvas.SetZIndex(newEndDistantLine, 0); |
|  | Canvas.SetZIndex(newBeginDistantLine, 0); |
|  | } |
|  | // Если перетаскивали окно, создаем его шаблон |
|  | else if (str[0] == "System.Windows.Controls.Border") |
|  | { |
|  | Border border = (Border)e.Data.GetData("System.Windows.Controls.Border"); |
|  | Border newBorder = copyBorder(border, e.GetPosition(this)); |
|  | myWindows.Add(new Window(newBorder)); |
|  | drawPanel.Children.Add(newBorder); |
|  | Canvas.SetZIndex(newBorder, 3); |
|  | Canvas.SetZIndex(newBorder.Child, 2); |
|  | } |
|  | } |
|  |  |
|  | // Добавление эффект DragAndDrop, при перетаскивании объекта на поле для рисования |
|  | private void drawPanel\_DragEnter(object sender, DragEventArgs e) |
|  | { |
|  | if (e.Data.GetDataPresent(DataFormats.Text)) |
|  | { |
|  | e.Effects = DragDropEffects.Copy; |
|  | } |
|  | } |
|  |  |
|  | // Добавление стрелки в буфер обмена DragAndDrop |
|  | private void arrow\_canvas\_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e) |
|  | { |
|  | Canvas canvas = (Canvas)sender; |
|  | Line line = null; |
|  | foreach (Line obj in canvas.Children.OfType<Line>()) |
|  | { |
|  | line = obj; |
|  | break; |
|  | } |
|  |  |
|  | DragDrop.DoDragDrop(canvas, line, DragDropEffects.Copy); |
|  | } |
|  |  |
|  | // Событие создания нового текстового поля (аттрибута) в окне, при нажатии |
|  | // кнопки Enter внутри самого верхнего текстового поля (аттрибута) |
|  | public void mainTextBox\_PreviewKeyDown(object sender, KeyEventArgs e) |
|  | { |
|  | if (e.Key == Key.Enter) |
|  | { |
|  | TextBox textBox = (TextBox)sender; |
|  | StackPanel stackPanel = (StackPanel)textBox.Parent; |
|  | TextBox newTextBox = new TextBox(); |
|  | newTextBox.Text = ""; |
|  | newTextBox.PreviewKeyDown += new KeyEventHandler(TextBox\_PreviewKeyDown); |
|  | stackPanel.Children.Add(newTextBox); |
|  | newTextBox.Focus(); |
|  | } |
|  | } |
|  |  |
|  | // Событие создания нового текстового поля (аттрибута) или удаление текущего в окне, при нажатии |
|  | // кнопки Enter или delete, backspace cсоответственно внутри не самого верхнего текстового поля (аттрибута) |
|  | public void TextBox\_PreviewKeyDown(object sender, KeyEventArgs e) |
|  | { |
|  | if (e.Key == Key.Enter) |
|  | { |
|  | TextBox textBox = (TextBox)sender; |
|  | StackPanel stackPanel = (StackPanel)textBox.Parent; |
|  | TextBox newTextBox = new TextBox(); |
|  | newTextBox.Text = ""; |
|  | newTextBox.PreviewKeyDown += new KeyEventHandler(TextBox\_PreviewKeyDown); |
|  | stackPanel.Children.Add(newTextBox); |
|  | newTextBox.Focus(); |
|  | } |
|  | else if ((e.Key == Key.Back || e.Key == Key.Delete) && ((TextBox)sender).Text.Length == 0) |
|  | { |
|  | Console.WriteLine("key Back"); |
|  | TextBox textBox = (TextBox)sender; |
|  | textBox.Text = "hi!"; |
|  | StackPanel stackPanel = (StackPanel)textBox.Parent; |
|  | stackPanel.Children.Remove(textBox); |
|  | } |
|  | } |
|  |  |
|  | // Добавление окна в буфер обмена DragAndDrop |
|  | private void StackPanel\_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e) |
|  | { |
|  | Canvas canvas = (Canvas)sender; |
|  | Border border = null; |
|  | foreach (Border obj in canvas.Children.OfType<Border>()) |
|  | { |
|  | border = obj; |
|  | break; |
|  | } |
|  | DragDrop.DoDragDrop(canvas, border, DragDropEffects.Copy); |
|  | } |
|  |  |
|  | // Событие очистки поля рисования от стрелок и окон |
|  | private void Button\_MouseDown(object sender, MouseButtonEventArgs e) |
|  | { |
|  | drawPanel.Children.Clear(); |
|  | myArrows.Clear(); |
|  | myWindows.Clear(); |
|  | } |
|  |  |
|  | // метод создания нового окна |
|  | private Border copyBorder(Border border, Point point) |
|  | { |
|  | Border newBorder = new Border(); |
|  | StackPanel stackPanel = (StackPanel)border.Child; |
|  | newBorder.BorderThickness = border.BorderThickness; |
|  | newBorder.Height = border.Height; |
|  | newBorder.Width = border.Width; |
|  | newBorder.BorderBrush = border.BorderBrush; |
|  | newBorder.Background = Brushes.White; |
|  | newBorder.CornerRadius = border.CornerRadius; |
|  |  |
|  | StackPanel newStackPanel = new StackPanel(); |
|  | newStackPanel.Margin = new Thickness(0, 0, 0, 0); |
|  |  |
|  | TextBox newNameTextBox = new TextBox(); |
|  | newNameTextBox.Text = "Название"; |
|  | newNameTextBox.BorderBrush = Brushes.Black; |
|  | newNameTextBox.BorderThickness = new Thickness(0, 0, 0, 1); |
|  | newNameTextBox.FontWeight = FontWeights.Bold; |
|  | newNameTextBox.Background = Brushes.Transparent; |
|  | newNameTextBox.Padding = new Thickness(10, 10, 0, 0); |
|  | newNameTextBox.FontSize = 14; |
|  | newNameTextBox.FontFamily = new FontFamily("Times New Roman"); |
|  | newStackPanel.Children.Add(newNameTextBox); |
|  |  |
|  | TextBox newMainNameAttributeTextBox = new TextBox(); |
|  | newMainNameAttributeTextBox.Text = "Атрибут"; |
|  | newMainNameAttributeTextBox.VerticalAlignment = VerticalAlignment.Center; |
|  | newMainNameAttributeTextBox.Padding = new Thickness(5, 5, 0, 0); |
|  | newMainNameAttributeTextBox.BorderBrush = Brushes.Black; |
|  | newMainNameAttributeTextBox.FontSize = 14; |
|  | newMainNameAttributeTextBox.BorderThickness = new Thickness(0, 0, 0, 0); |
|  | newMainNameAttributeTextBox.FontFamily = new FontFamily("Times New Roman"); |
|  | newMainNameAttributeTextBox.PreviewKeyDown += new KeyEventHandler(mainTextBox\_PreviewKeyDown); |
|  | newStackPanel.Children.Add(newMainNameAttributeTextBox); |
|  |  |
|  | newBorder.Child = newStackPanel; |
|  | newBorder.Margin = new Thickness(point.X - Config.getMarginLeftPanel(), point.Y - Config.getMarginTopPanel(), 0, 0); |
|  | return newBorder; |
|  | } |
|  |  |
|  | // Событие сохранения резулатата работы путем создания скриншота |
|  | private async void Button\_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e) |
|  | { |
|  | string filePath = "./Screenshot.png"; |
|  | RenderTargetBitmap renderTargetBitmap = |
|  | new RenderTargetBitmap(1920, 1080, 0, 0, PixelFormats.Pbgra32); |
|  | renderTargetBitmap.Render(drawPanel); |
|  | PngBitmapEncoder pngImage = new PngBitmapEncoder(); |
|  | pngImage.Frames.Add(BitmapFrame.Create(renderTargetBitmap)); |
|  | using (Stream fileStream = File.Create(filePath)) |
|  | { |
|  | pngImage.Save(fileStream); |
|  | } |
|  |  |
|  | ToolTip tooltip = new ToolTip(); |
|  | Button button = (Button)sender; |
|  | tooltip.Content = "Сохранено"; |
|  | button.ToolTip = tooltip; |
|  | tooltip.IsOpen = true; |
|  | await Task.Delay(2000); |
|  | tooltip.IsOpen = false; |
|  | } |
|  |  |
|  | // Событие удаления стрелки или окон, при перетаскивании их за пределы поля для рисования |
|  | private void drawPanel\_MouseLeave(object sender, MouseEventArgs e) |
|  | { |
|  | if (indexCurrentWindow != -1) |
|  | { |
|  | myWindows[indexCurrentWindow].remove(); |
|  | myWindows.Remove(myWindows[indexCurrentWindow]); |
|  | indexCurrentWindow = -1; |
|  | } |
|  | else if (indexCurrentArrow != -1) |
|  | { |
|  | myArrows[indexCurrentArrow].remove(); |
|  | myArrows.Remove(myArrows[indexCurrentArrow]); |
|  | indexCurrentArrow = -1; |
|  | } |
|  | } |
|  | } |
|  | } |

**Файл Arrow.cs**

|  |  |
| --- | --- |
| using System; |  |
|  |  | using System.Collections.Generic; |
|  |  | using System.Linq; |
|  |  | using System.Text; |
|  |  | using System.Threading.Tasks; |
|  |  | using System.Windows.Shapes; |
|  |  | using System.Windows.Navigation; |
|  |  | using System.Windows; |
|  |  | using System.Windows.Media; |
|  |  |  |
|  |  | namespace Editor |
|  |  | { |
|  |  | // пользовательский класс стрелки (несколько линий и ломанных) |
|  |  | abstract public class Arrow |
|  |  | { |
|  |  | // главная линия |
|  |  | protected Line line; |
|  |  | // дальняя от конца перекрестная линия |
|  |  | protected Line endDistantLine; |
|  |  | // дальняя от начала перекрестная линия |
|  |  | protected Line beginDistantLine; |
|  |  |  |
|  |  | // конструктор класса |
|  |  | public Arrow() { } |
|  |  |  |
|  |  | // метод, меняющий позицию начальной точки |
|  |  | public abstract void setPoint1(double x1, double y1); |
|  |  | // метод, меняющий позицию начальной точки |
|  |  | public abstract void setPoint1(Point point); |
|  |  | // метод, меняющий позицию конечной точки |
|  |  | public abstract void setPoint2(double x2, double y2); |
|  |  | // метод, меняющий позицию конечной точки |
|  |  | public abstract void setPoint2(Point point); |
|  |  | // метод, удаляющий все линиии и ломанные из поля для рисования |
|  |  | public abstract void remove(); |
|  |  | // геттер точки, указывающий на позицию начала стрелки |
|  |  | public Point getPoint1() |
|  |  | { |
|  |  | return new Point(this.line.X1, this.line.Y1); |
|  |  | } |
|  |  | // геттер точки, указывающий на позицию конца стрелки |
|  |  | public Point getPoint2() |
|  |  | { |
|  |  | return new Point(this.line.X2, this.line.Y2); |
|  |  | } |
|  |  |  |
|  |  | // метод сравнивающий строку с именем линии |
|  |  | public bool equalName(string name) |
|  |  | { |
|  |  | return this.line.Name == name; |
|  |  | } |
|  |  | // геттер имени |
|  |  | public string name() |
|  |  | { |
|  |  | return this.line.Name; |
|  |  | } |
|  |  |  |
|  |  | // проверяет, находится ли стрелка около точки. Если находится, то возвращает |
|  |  | // 1 - рядом с началом стрелки, 2 - рядом с концом стрелки, 0 - около точки нет стрелки |
|  |  | public int equalLineWithPoint(Point point) |
|  |  | { |
|  |  | int pointLineIndex = 0; |
|  |  | if (Math.Abs(point.X - this.line.X2 - Config.getMarginLeftPanel()) < Config.getRadius() / 2 && Math.Abs(point.Y - this.line.Y2 - Config.getMarginTopPanel()) < Config.getRadius() / 2) |
|  |  | { |
|  |  | pointLineIndex = 2; |
|  |  | } |
|  |  | else if (Math.Abs(point.X - this.line.X1 - Config.getMarginLeftPanel()) < Config.getRadius() / 2 && Math.Abs(point.Y - this.line.Y1 - Config.getMarginTopPanel()) < Config.getRadius() / 2) |
|  |  | { |
|  |  | pointLineIndex = 1; |
|  |  | } |
|  |  |  |
|  |  | return pointLineIndex; |
|  |  | } |
|  |  |  |
|  |  | // статический метод, соединяющий две стрелки |
|  |  | public static bool concateLines(Arrow firstLine, Arrow secondLine) |
|  |  | { |
|  |  | bool result = false; |
|  |  | if (Math.Abs(firstLine.getPoint2().X - secondLine.getPoint2().X) < Config.getRadius() && Math.Abs(firstLine.getPoint2().Y - secondLine.getPoint2().Y) < Config.getRadius()) |
|  |  | { |
|  |  | firstLine.setPoint2(secondLine.getPoint2().X + Config.getMarginLeftPanel(), secondLine.getPoint2().Y + Config.getMarginTopPanel()); |
|  |  | result = true; |
|  |  | } |
|  |  | else if (Math.Abs(firstLine.getPoint2().X - secondLine.getPoint1().X) < Config.getRadius() && Math.Abs(firstLine.getPoint2().Y - secondLine.getPoint1().Y) < Config.getRadius()) |
|  |  | { |
|  |  | firstLine.setPoint2(secondLine.getPoint1().X + Config.getMarginLeftPanel(), secondLine.getPoint1().Y + Config.getMarginTopPanel()); |
|  |  | result = true; |
|  |  | } |
|  |  | else if (Math.Abs(firstLine.getPoint1().X - secondLine.getPoint1().X) < Config.getRadius() && Math.Abs(firstLine.getPoint1().Y - secondLine.getPoint1().Y) < Config.getRadius()) |
|  |  | { |
|  |  | firstLine.setPoint1(secondLine.getPoint1().X + Config.getMarginLeftPanel(), secondLine.getPoint1().Y + Config.getMarginTopPanel()); |
|  |  | result = true; |
|  |  | } |
|  |  | else if (Math.Abs(firstLine.getPoint1().X - secondLine.getPoint2().X) < Config.getRadius() && Math.Abs(firstLine.getPoint1().Y - secondLine.getPoint2().Y) < Config.getRadius()) |
|  |  | { |
|  |  | firstLine.setPoint1(secondLine.getPoint2().X + Config.getMarginLeftPanel(), secondLine.getPoint2().Y + Config.getMarginTopPanel()); |
|  |  | result = true; |
|  |  | } |
|  |  | return result; |
|  |  | } |
|  |  |  |
|  |  | // Копирует главную линию |
|  |  | public static Line copyLine(Line line, string name, Point point) |
|  |  | { |
|  |  | Line newLine = new Line(); |
|  |  | newLine.X1 = point.X - Config.getMarginLeftPanel(); |
|  |  | newLine.Y1 = point.Y - Config.getMarginTopPanel(); |
|  |  | newLine.X2 = newLine.X1 + Math.Abs(line.X2 - line.X1); |
|  |  | newLine.Y2 = newLine.Y1 + Math.Abs(line.Y2 - line.Y1); |
|  |  | newLine.Stroke = line.Stroke; |
|  |  | newLine.StrokeThickness = line.StrokeThickness; |
|  |  | newLine.StrokeDashArray = line.StrokeDashArray; |
|  |  | newLine.Name = name; |
|  |  | return newLine; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий дальнюю от конца линию по шаблону |
|  |  | public static Line patternEndDistantLine(Line line) |
|  |  | { |
|  |  | Line newDistantLine = new Line(); |
|  |  | newDistantLine.X1 = line.X2 - 20; |
|  |  | newDistantLine.Y1 = line.Y2 - 10; |
|  |  | newDistantLine.X2 = line.X2 - 20; |
|  |  | newDistantLine.Y2 = line.Y2 + 10; |
|  |  | newDistantLine.Stroke = Brushes.Black; |
|  |  | newDistantLine.StrokeThickness = 1; |
|  |  | return newDistantLine; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий дальнюю от начала линию по шаблону |
|  |  | public static Line patternBeginDistantLine(Line line) |
|  |  | { |
|  |  | Line newDistantLine = new Line(); |
|  |  | newDistantLine.X1 = line.X1 + 20; |
|  |  | newDistantLine.Y1 = line.Y1 + 10; |
|  |  | newDistantLine.X2 = line.X1 + 20; |
|  |  | newDistantLine.Y2 = line.Y1 - 10; |
|  |  | newDistantLine.Stroke = Brushes.Black; |
|  |  | newDistantLine.StrokeThickness = 1; |
|  |  | return newDistantLine; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий ближнюю от конца линию по шаблону |
|  |  | public static Line patternEndNearLine(Line line) |
|  |  | { |
|  |  | Line newNearLine = new Line(); |
|  |  | newNearLine.X1 = line.X2 - 10; |
|  |  | newNearLine.Y1 = line.Y2 - 10; |
|  |  | newNearLine.X2 = line.X2 - 10; |
|  |  | newNearLine.Y2 = line.Y2 + 10; |
|  |  | newNearLine.Stroke = Brushes.Black; |
|  |  | newNearLine.StrokeThickness = 1; |
|  |  | return newNearLine; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий ближнюю от начала линию по шаблону |
|  |  | public static Line patternBeginNearLine(Line line) |
|  |  | { |
|  |  | Line newNearLine = new Line(); |
|  |  | newNearLine.X1 = line.X1 + 10; |
|  |  | newNearLine.Y1 = line.Y1 + 10; |
|  |  | newNearLine.X2 = line.X1 + 10; |
|  |  | newNearLine.Y2 = line.Y1 - 10; |
|  |  | newNearLine.Stroke = Brushes.Black; |
|  |  | newNearLine.StrokeThickness = 1; |
|  |  | return newNearLine; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий ломанную линию на конце стрелки по шаблону |
|  |  | public static Polyline patternEndPolyline(Line line) |
|  |  | { |
|  |  | Polyline newPolyline = new Polyline(); |
|  |  | newPolyline.Points.Add(new Point(line.X2, line.Y2 - 10)); |
|  |  | newPolyline.Points.Add(new Point(line.X2 - 20, line.Y2)); |
|  |  | newPolyline.Points.Add(new Point(line.X2, line.Y2 + 10)); |
|  |  |  |
|  |  | newPolyline.Stroke = Brushes.Black; |
|  |  | newPolyline.StrokeThickness = 1; |
|  |  | return newPolyline; |
|  |  | } |
|  |  |  |
|  |  | // метод, создающий ломанную линию в начале стрелки по шаблону |
|  |  | public static Polyline patternBeginPolyline(Line line) |
|  |  | { |
|  |  | Polyline newPolyline = new Polyline(); |
|  |  | newPolyline.Points.Add(new Point(line.X1, line.Y2 - 10)); |
|  |  | newPolyline.Points.Add(new Point(line.X1 + 20, line.Y2)); |
|  |  | newPolyline.Points.Add(new Point(line.X1, line.Y2 + 10)); |
|  |  |  |
|  |  | newPolyline.Stroke = Brushes.Black; |
|  |  | newPolyline.StrokeThickness = 1; |
|  |  | return newPolyline; |
|  |  | } |
|  |  | } |
|  |  | }  **Файл Arrow1.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.Shapes; | |  |  | |  | namespace Editor | |  | { | |  | // пользовательский класс стрелки первого типа | |  | class Arrow1 : Arrow | |  | { | |  | // ближняя линия от конца стрелки | |  | private Line endNearLine; | |  | // ближняя линия от начала стрелки | |  | private Line beginNearLine; | |  |  | |  | // конструктор класса | |  | public Arrow1(Line line, Line endDistantLine, Line beginDistantLine, Line endNearLine, Line beginNearLine) | |  | { | |  | this.line = line; | |  | this.endDistantLine = endDistantLine; | |  | this.beginDistantLine = beginDistantLine; | |  | this.endNearLine = endNearLine; | |  | this.beginNearLine = beginNearLine; | |  | } | |  |  | |  | public override void setPoint1(double x1, double y1) | |  | { | |  | this.line.X1 = x1 - Config.getMarginLeftPanel(); | |  | this.line.Y1 = y1 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint1(Point point) | |  | { | |  | this.line.X1 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y1 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(double x2, double y2) | |  | { | |  | this.line.X2 = x2 - Config.getMarginLeftPanel(); | |  | this.line.Y2 = y2 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(Point point) | |  | { | |  | this.line.X2 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y2 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | // метод, меняющий расположение стрелок на концах | |  | private void moveArrow() | |  | { | |  | double d = Math.Sqrt(Math.Pow(this.getPoint2().X - this.getPoint1().X, 2) + Math.Pow(this.getPoint2().Y - this.getPoint1().Y, 2)); | |  |  | |  | double X = this.getPoint2().X - this.getPoint1().X; | |  | double Y = this.getPoint2().Y - this.getPoint1().Y; | |  |  | |  | double X3 = this.getPoint2().X - (X / d) \* 10; | |  | double Y3 = this.getPoint2().Y - (Y / d) \* 10; | |  |  | |  | double Xp = this.getPoint2().Y - this.getPoint1().Y; | |  | double Yp = this.getPoint1().X - this.getPoint2().X; | |  |  | |  | double X4 = X3 + (Xp / d) \* 11; | |  | double Y4 = Y3 + (Yp / d) \* 11; | |  | double X5 = X3 - (Xp / d) \* 11; | |  | double Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.endNearLine.X1 = X4; | |  | this.endNearLine.Y1 = Y4; | |  |  | |  | this.endNearLine.X2 = X5; | |  | this.endNearLine.Y2 = Y5; | |  |  | |  | X3 = this.getPoint2().X - (X / d) \* 20; | |  | Y3 = this.getPoint2().Y - (Y / d) \* 20; | |  |  | |  | Xp = this.getPoint2().Y - this.getPoint1().Y; | |  | Yp = this.getPoint1().X - this.getPoint2().X; | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.endDistantLine.X1 = X4; | |  | this.endDistantLine.Y1 = Y4; | |  |  | |  | this.endDistantLine.X2 = X5; | |  | this.endDistantLine.Y2 = Y5; | |  |  | |  | X = -1 \* X; | |  | Y = -1 \* Y; | |  |  | |  | X3 = this.getPoint1().X - (X / d) \* 10; | |  | Y3 = this.getPoint1().Y - (Y / d) \* 10; | |  |  | |  | Xp = -1 \* Xp; | |  | Yp = -1 \* Yp; | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.beginNearLine.X1 = X4; | |  | this.beginNearLine.Y1 = Y4; | |  |  | |  | this.beginNearLine.X2 = X5; | |  | this.beginNearLine.Y2 = Y5; | |  |  | |  |  | |  | X3 = this.getPoint1().X - (X / d) \* 20; | |  | Y3 = this.getPoint1().Y - (Y / d) \* 20; | |  |  | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.beginDistantLine.X1 = X4; | |  | this.beginDistantLine.Y1 = Y4; | |  |  | |  | this.beginDistantLine.X2 = X5; | |  | this.beginDistantLine.Y2 = Y5; | |  | } | |  |  | |  | override public void remove() | |  | { | |  | Canvas canvas = (Canvas)this.line.Parent; | |  | canvas.Children.Remove(this.line); | |  | canvas.Children.Remove(this.endDistantLine); | |  | canvas.Children.Remove(this.beginDistantLine); | |  | canvas.Children.Remove(this.endNearLine); | |  | canvas.Children.Remove(this.beginNearLine); | |  | } | |  | } | |  | } |   **Файл Arrow2.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.Shapes; | |  |  | |  | namespace Editor | |  | { | |  | // пользовательский класс стрелки второго типа | |  | class Arrow2 : Arrow | |  | { | |  | // ломанная линия на конце стрелки | |  | private Polyline endPolyline; | |  | // ближняя линия от начала стрелки | |  | private Line beginNearLine; | |  |  | |  | // конструктор класса | |  | public Arrow2(Line line, Line endDistantLine, Line beginDistantLine, Line beginNearLine, Polyline endPolyline) | |  | { | |  | this.line = line; | |  | this.endDistantLine = endDistantLine; | |  | this.beginDistantLine = beginDistantLine; | |  | this.beginNearLine = beginNearLine; | |  | this.endPolyline = endPolyline; | |  | } | |  |  | |  | public override void setPoint1(double x1, double y1) | |  | { | |  | this.line.X1 = x1 - Config.getMarginLeftPanel(); | |  | this.line.Y1 = y1 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint1(Point point) | |  | { | |  | this.line.X1 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y1 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(double x2, double y2) | |  | { | |  | this.line.X2 = x2 - Config.getMarginLeftPanel(); | |  | this.line.Y2 = y2 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(Point point) | |  | { | |  | this.line.X2 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y2 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | // метод, меняющий расположение стрелок на концах | |  | private void moveArrow() | |  | { | |  | endPolyline.Points.Clear(); | |  |  | |  | double d = Math.Sqrt(Math.Pow(this.getPoint2().X - this.getPoint1().X, 2) + Math.Pow(this.getPoint2().Y - this.getPoint1().Y, 2)); | |  |  | |  | double X = this.getPoint2().X - this.getPoint1().X; | |  | double Y = this.getPoint2().Y - this.getPoint1().Y; | |  |  | |  | double X3 = this.getPoint2().X - (X / d) \* 20; | |  | double Y3 = this.getPoint2().Y - (Y / d) \* 20; | |  |  | |  | double Xp = this.getPoint2().Y - this.getPoint1().Y; | |  | double Yp = this.getPoint1().X - this.getPoint2().X; | |  |  | |  | double X4 = X3 + (Xp / d) \* 11; | |  | double Y4 = Y3 + (Yp / d) \* 11; | |  | double X5 = X3 - (Xp / d) \* 11; | |  | double Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.endDistantLine.X1 = X4; | |  | this.endDistantLine.Y1 = Y4; | |  |  | |  | this.endDistantLine.X2 = X5; | |  | this.endDistantLine.Y2 = Y5; | |  |  | |  | X4 = this.getPoint2().X + (Xp / d) \* 11; | |  | Y4 = this.getPoint2().Y + (Yp / d) \* 11; | |  |  | |  | X5 = this.getPoint2().X - (Xp / d) \* 11; | |  | Y5 = this.getPoint2().Y - (Yp / d) \* 11; | |  |  | |  | this.endPolyline.Points.Add(new Point(X4, Y4)); | |  | this.endPolyline.Points.Add(new Point(X3, Y3)); | |  | this.endPolyline.Points.Add(new Point(X5, Y5)); | |  |  | |  | X = -1 \* X; | |  | Y = -1 \* Y; | |  |  | |  | X3 = this.getPoint1().X - (X / d) \* 10; | |  | Y3 = this.getPoint1().Y - (Y / d) \* 10; | |  |  | |  | Xp = -1 \* Xp; | |  | Yp = -1 \* Yp; | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.beginNearLine.X1 = X4; | |  | this.beginNearLine.Y1 = Y4; | |  |  | |  | this.beginNearLine.X2 = X5; | |  | this.beginNearLine.Y2 = Y5; | |  |  | |  |  | |  | X3 = this.getPoint1().X - (X / d) \* 20; | |  | Y3 = this.getPoint1().Y - (Y / d) \* 20; | |  |  | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.beginDistantLine.X1 = X4; | |  | this.beginDistantLine.Y1 = Y4; | |  |  | |  | this.beginDistantLine.X2 = X5; | |  | this.beginDistantLine.Y2 = Y5; | |  | } | |  |  | |  | override public void remove() | |  | { | |  | Canvas canvas = (Canvas)this.line.Parent; | |  | canvas.Children.Remove(this.line); | |  | canvas.Children.Remove(this.endDistantLine); | |  | canvas.Children.Remove(this.beginDistantLine); | |  | canvas.Children.Remove(this.endPolyline); | |  | canvas.Children.Remove(this.beginNearLine); | |  | } | |  | } | |  | } |   **Файл Arrow3.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.Shapes; | |  |  | |  | namespace Editor | |  | { | |  | // пользовательский класс стрелки третьего типа | |  | class Arrow3 : Arrow | |  | { | |  | // ломанная линия на конце стрелки | |  | private Polyline endPolyline; | |  | // ломанная линия в начале стрелки | |  | private Polyline beginPolyline; | |  |  | |  | // конструктор класса | |  | public Arrow3(Line line, Line endDistantLine, Line beginDistantLine, Polyline endPolyline, Polyline beginPolyline) | |  | { | |  | this.line = line; | |  | this.endDistantLine = endDistantLine; | |  | this.beginDistantLine = beginDistantLine; | |  | this.endPolyline = endPolyline; | |  | this.beginPolyline = beginPolyline; | |  | } | |  |  | |  | public override void setPoint1(double x1, double y1) | |  | { | |  | this.line.X1 = x1 - Config.getMarginLeftPanel(); | |  | this.line.Y1 = y1 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint1(Point point) | |  | { | |  | this.line.X1 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y1 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(double x2, double y2) | |  | { | |  | this.line.X2 = x2 - Config.getMarginLeftPanel(); | |  | this.line.Y2 = y2 - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | public override void setPoint2(Point point) | |  | { | |  | this.line.X2 = point.X - Config.getMarginLeftPanel(); | |  | this.line.Y2 = point.Y - Config.getMarginTopPanel(); | |  | this.moveArrow(); | |  | } | |  |  | |  | // метод, меняющий расположение стрелок на концах | |  | private void moveArrow() | |  | { | |  | this.endPolyline.Points.Clear(); | |  | this.beginPolyline.Points.Clear(); | |  |  | |  | double d = Math.Sqrt(Math.Pow(this.getPoint2().X - this.getPoint1().X, 2) + Math.Pow(this.getPoint2().Y - this.getPoint1().Y, 2)); | |  |  | |  | double X = this.getPoint2().X - this.getPoint1().X; | |  | double Y = this.getPoint2().Y - this.getPoint1().Y; | |  |  | |  | double X3 = this.getPoint2().X - (X / d) \* 20; | |  | double Y3 = this.getPoint2().Y - (Y / d) \* 20; | |  |  | |  | double Xp = this.getPoint2().Y - this.getPoint1().Y; | |  | double Yp = this.getPoint1().X - this.getPoint2().X; | |  |  | |  | double X4 = X3 + (Xp / d) \* 11; | |  | double Y4 = Y3 + (Yp / d) \* 11; | |  | double X5 = X3 - (Xp / d) \* 11; | |  | double Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.endDistantLine.X1 = X4; | |  | this.endDistantLine.Y1 = Y4; | |  |  | |  | this.endDistantLine.X2 = X5; | |  | this.endDistantLine.Y2 = Y5; | |  |  | |  | X4 = this.getPoint2().X + (Xp / d) \* 11; | |  | Y4 = this.getPoint2().Y + (Yp / d) \* 11; | |  |  | |  | X5 = this.getPoint2().X - (Xp / d) \* 11; | |  | Y5 = this.getPoint2().Y - (Yp / d) \* 11; | |  |  | |  | this.endPolyline.Points.Add(new Point(X4, Y4)); | |  | this.endPolyline.Points.Add(new Point(X3, Y3)); | |  | this.endPolyline.Points.Add(new Point(X5, Y5)); | |  |  | |  | X = -1 \* X; | |  | Y = -1 \* Y; | |  |  | |  | X3 = this.getPoint1().X - (X / d) \* 20; | |  | Y3 = this.getPoint1().Y - (Y / d) \* 20; | |  |  | |  | Xp = -1 \* Xp; | |  | Yp = -1 \* Yp; | |  |  | |  | X4 = X3 + (Xp / d) \* 11; | |  | Y4 = Y3 + (Yp / d) \* 11; | |  | X5 = X3 - (Xp / d) \* 11; | |  | Y5 = Y3 - (Yp / d) \* 11; | |  |  | |  | this.beginDistantLine.X1 = X4; | |  | this.beginDistantLine.Y1 = Y4; | |  | this.beginDistantLine.X2 = X5; | |  | this.beginDistantLine.Y2 = Y5; | |  |  | |  | X4 = this.getPoint1().X + (Xp / d) \* 11; | |  | Y4 = this.getPoint1().Y + (Yp / d) \* 11; | |  |  | |  | X5 = this.getPoint1().X - (Xp / d) \* 11; | |  | Y5 = this.getPoint1().Y - (Yp / d) \* 11; | |  |  | |  | this.beginPolyline.Points.Add(new Point(X4, Y4)); | |  | this.beginPolyline.Points.Add(new Point(X3, Y3)); | |  | this.beginPolyline.Points.Add(new Point(X5, Y5)); | |  | } | |  |  | |  |  | |  | override public void remove() | |  | { | |  | Canvas canvas = (Canvas)this.line.Parent; | |  | canvas.Children.Remove(this.line); | |  | canvas.Children.Remove(this.endDistantLine); | |  | canvas.Children.Remove(this.beginDistantLine); | |  | canvas.Children.Remove(this.endPolyline); | |  | canvas.Children.Remove(this.beginPolyline); | |  | } | |  | } | |  | } |   **Файл Config.cs**   |  | | --- | | using System; | |  | using System.Collections.Generic; | |  | using System.Linq; | |  | using System.Text; | |  | using System.Threading.Tasks; | |  |  | |  | namespace Editor | |  | { | |  | // класс конфигурации приложения | |  | class Config | |  | { | |  | // расстояние до точек, при котором будут соединяться линии к другим линиям или к окну. | |  | private static double rad; | |  | // расстояние от панели рисования до верхней границы окна. | |  | private static double toppanel; | |  | // расстояние от панели рисования до левой границы окна. | |  | private static double leftpanel; | |  |  | |  | // Конструктор классса (приватный, чтобы не создать экземпляр класса) | |  | private Config() | |  | { | |  | } | |  |  | |  | // геттер | |  | public static double getRadius() | |  | { | |  | return ra; | |  | } | |  |  | |  | // геттер | |  | public static double getMarginTopPanel() | |  | { | |  | return toppanel; | |  | } | |  |  | |  | // геттер | |  | public static double getMarginLeftPanel() | |  | { | |  | return leftpanel; | |  | } | |  |  | |  | // сеттер | |  | public static void setConfig(double newRadius, double newMarginTopPanel, double newMarginLeftPanel) | |  | { | |  | rad = newRadius; | |  | toppanel = newMarginTopPanel; | |  | leftpanel = newMarginLeftPanel; | |  | } | |  | } | |  | } |   **Файл LinkArr.cs**   |  | | --- | | using System; | |  | using System.Collections.Generic; | |  | using System.Linq; | |  | using System.Text; | |  | using System.Threading.Tasks; | |  |  | |  | namespace Editor | |  | { | |  | // класс, связывающий стрелки с окном | |  | class LinkArr | |  | { | |  | // указывает какая сторона стрелки связана с окном | |  | bool indexSide; | |  | // указывает какая стрлка связана с окном | |  | Arrow line; | |  | // указывает какая сторона окна связана со стрелкой | |  | char location; | |  |  | |  | // конструктор класса | |  | public LinkArr(bool indexSide, Arrow line, char location) | |  | { | |  | this.indexSide = indexSide; | |  | this.line = line; | |  | this.location = location; | |  | } | |  |  | |  | // геттер | |  | public bool getIndesSide() | |  | { | |  | return this.indexSide; | |  | } | |  |  | |  | // геттер | |  | public Arrow getLine() | |  | { | |  | return this.line; | |  | } | |  |  | |  | // геттер | |  | public char getLocation() | |  | { | |  | return this.location; | |  | } | |  | } | |  | } |   **Файл Window.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.Input; | |  | using System.Windows.Media; | |  |  | |  | namespace Editor | |  | { | |  | // пользовательский класс окна | |  | class Window | |  | { | |  | // координаты точки внутри окна при перемещении этого окна | |  | private Point offset; | |  | // рамка окна | |  | private Border border; | |  | // связанные с окном стрелки | |  | private List<LinkArr> linkArrows; | |  |  | |  | // конструктор класса | |  | public Window(Border border) | |  | { | |  | this.border = border; | |  | offset = new Point(); | |  | linkArrows = new List<LinkArr>(); | |  | } | |  |  | |  | // координата располодения окна слева (отступ) | |  | public double getMarginLeft() | |  | { | |  | return this.border.Margin.Left; | |  | } | |  |  | |  | // координата располодения окна сверху (отступ) | |  | public double getMarginTop() | |  | { | |  | return this.border.Margin.Top; | |  | } | |  |  | |  | // ширина окна | |  | public double getWidth() | |  | { | |  | return this.border.Width; | |  | } | |  |  | |  | // метод, соединяющий окно с линией | |  | public bool concateWindowWithLine(Arrow line) | |  | { | |  | bool result = false; | |  | if (Math.Abs(line.getPoint2().X - (this.border.Margin.Left + this.border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - this.border.Margin.Top) < Config.getRadius()) | |  | { | |  | line.setPoint2(this.border.Margin.Left + this.border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint2().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height)) < Config.getRadius()) | |  | { | |  | line.setPoint2(this.border.Margin.Left + this.border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + this.border.Height + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - border.Margin.Top) < Config.getRadius()) | |  | { | |  | line.setPoint1(this.border.Margin.Left + this.border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height)) < Config.getRadius()) | |  | { | |  | line.setPoint1(this.border.Margin.Left + border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint1().X - border.Margin.Left) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | line.setPoint1(this.border.Margin.Left + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | line.setPoint1(this.border.Margin.Left + border.Width + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint2().X - border.Margin.Left) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | line.setPoint2(this.border.Margin.Left + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  | else if (Math.Abs(line.getPoint2().X - (border.Margin.Left + border.Width)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | line.setPoint2(this.border.Margin.Left + border.Width + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | result = true; | |  | } | |  |  | |  | return result; | |  | } | |  |  | |  | // метод, добавляющий связанные с окном стрелки в массив | |  | public void addLinkingArrowsInList(Arrow line) | |  | { | |  | if (Math.Abs(line.getPoint2().X - (this.border.Margin.Left + this.border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - this.border.Margin.Top) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(false, line, 'T')); | |  | } | |  | else if (Math.Abs(line.getPoint2().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(false, line, 'B')); | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - border.Margin.Top) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(true, line, 'T')); | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width / 2)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(true, line, 'B')); | |  | } | |  | else if (Math.Abs(line.getPoint1().X - border.Margin.Left) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(true, line, 'L')); | |  | } | |  | else if (Math.Abs(line.getPoint1().X - (border.Margin.Left + border.Width)) < Config.getRadius() && Math.Abs(line.getPoint1().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(true, line, 'R')); | |  | } | |  | else if (Math.Abs(line.getPoint2().X - border.Margin.Left) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(false, line, 'L')); | |  | } | |  | else if (Math.Abs(line.getPoint2().X - (border.Margin.Left + border.Width)) < Config.getRadius() && Math.Abs(line.getPoint2().Y - (border.Margin.Top + border.Height / 2)) < Config.getRadius()) | |  | { | |  | linkArrows.Add(new LinkArr(false, line, 'R')); | |  | } | |  | } | |  |  | |  | // метод, перемещающий связанные с окном стрелки | |  | private void moveArrows() | |  | { | |  | foreach (LinkArr link in linkArrows) | |  | { | |  | if (link.getIndesSide()) // начало стрелки | |  | { | |  | switch (link.getLocation()) | |  | { | |  | case 'T': | |  | link.getLine().setPoint1(this.border.Margin.Left + this.border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + Config.getMarginTopPanel()); | |  | break; | |  | case 'B': | |  | link.getLine().setPoint1(this.border.Margin.Left + border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height + Config.getMarginTopPanel()); | |  | break; | |  | case 'L': | |  | link.getLine().setPoint1(this.border.Margin.Left + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | break; | |  | case 'R': | |  | link.getLine().setPoint1(this.border.Margin.Left + border.Width + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | break; | |  | } | |  | } | |  | else // конец стрелки | |  | { | |  | switch (link.getLocation()) | |  | { | |  | case 'T': | |  | link.getLine().setPoint2(this.border.Margin.Left + this.border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + Config.getMarginTopPanel()); | |  | break; | |  | case 'B': | |  | link.getLine().setPoint2(this.border.Margin.Left + border.Width / 2 + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height + Config.getMarginTopPanel()); | |  | break; | |  | case 'L': | |  | link.getLine().setPoint2(this.border.Margin.Left + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | break; | |  | case 'R': | |  | link.getLine().setPoint2(this.border.Margin.Left + border.Width + Config.getMarginLeftPanel(), this.border.Margin.Top + border.Height / 2 + Config.getMarginTopPanel()); | |  | break; | |  | } | |  | } | |  | } | |  | } | |  |  | |  | // очищает массив связанных стрелок | |  | public void clearLinkArrows() | |  | { | |  | this.linkArrows.Clear(); | |  | } | |  |  | |  | // проверяет находится ли окно под курсором | |  | public bool equalWindowWithPoint(Point point) | |  | { | |  | bool result = false; | |  | if (point.X - Config.getMarginLeftPanel() >= this.border.Margin.Left && point.X - Config.getMarginLeftPanel() <= this.border.Margin.Left + this.border.Width && | |  | point.Y - Config.getMarginTopPanel() >= this.border.Margin.Top && point.Y - Config.getMarginTopPanel() <= this.border.Margin.Top + this.border.Height) | |  | { | |  | offset.X = point.X - this.border.Margin.Left; | |  | offset.Y = point.Y - this.border.Margin.Top; | |  | result = true; | |  | } | |  | return result; | |  | } | |  |  | |  | // метод, перемещающий окно в точку | |  | public void move(Point point) | |  | { | |  | this.border.Margin = new Thickness(point.X - offset.X, point.Y - offset.Y, 0, 0); | |  | this.moveArrows(); | |  | } | |  |  | |  | // метод, удаляющий окно с поля для рисования | |  | public void remove() | |  | { | |  | Canvas canvas = (Canvas)this.border.Parent; | |  | linkArrows.Clear(); | |  | canvas.Children.Remove(this.border); | |  | } | |  | } | |  | }  **Пример выполнения программы** | |



**Контрольные вопросы**

1. Зачем нужен перегруженный оператор присваивания?

Перегрузка оператора присваивания — это по сути процедура копирования величин из объекта в объект. Введение. Процедура перегрузки операторов, фактически, считается самым удобным способом задания функций. Использовать её нужно с целью упрощения представления кодов, но это не должно вызывать препятствий в их чтении. Необходимо учитывать, так же, невозможность перегрузки операторов совместно с их типами, если они включены.

1. Зачем нужен механизм наследования?

Для повторного использования одних и тех же методов и функций, чтобы сократить количество используемого кода.

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

Модификаторы наследования используются для контроля над модификаторами доступа создаваемого класса.   
− private (закрытый, доступный только собственным методам)

− public (открытый, доступный любым функциям)

− protected (защищенный, доступный только собственным методам и методам производных классов)

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, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий.

1. Что понимается под динамическим полиморфизмом?

Динамический полиморфизм– выбор функции производится по используемому при выводе имени, типу вызываемого объекта и типу используемый параметров. Если объявленный тип не совпадает с типом использованного объекта, тип этого объекта идентифицируется и проверяется во время исполнения программы, что позволяет осуществить правильный выбор соответствующей функции. Это и называется динамическим полиморфизмом (поздним связыванием).

1. Что такое интерфейс класса?

Интерфейс класса – система связи класса и пользователя.

1. Зачем нужен чисто виртуальный метод? Как он выглядит?

Чисто виртуальный метод - метод, который объявляется в базовом классе, но не имеет в нем определения. Виртуальный метод становится полезным только тогда, когда нужно программно один и тот же метод менять по некоторому алгоритму в программном коде. Поскольку он не имеет определения, то есть реализации, то всякий производный класс обязан иметь свою собственную версию определения.  
virtual (тип данных метода) (имя метода) (необходимые параметры) = 0;

1. Какой класс называется абстрактным?

Абстрактные классы в объектно-ориентированном программировании — это базовые классы, которые можно наследовать, но нельзя реализовывать. То есть на их основе нельзя создать объект.

**Вывод:** в ходе работы я закрепила навыки использования механизмов ООП на примере реализации графического редактора.