Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

**Межрегиональный учебный центр переподготовки специалистов**

**Курсовая работа**

по дисциплине: Объектно-ориентированное программирование

Тема 28: Создать простейший векторный графический редактор.

**Выполнил**: Коростелин А. В.

**Группа:** ПБТ-11

**Проверила**: Ситняковская Е. И.

Новосибирск, 2022

**Задание**

Написать программу, используя объектно-ориентированный подход. Тему выбираете самостоятельно. Описание классов желательно оформить в виде отдельного модуля. Иерархия классов должна включать минимум четыре класса, один из которых – абстрактный.

Язык и среда программирования – Ваш выбор. Можете выполнять работы на языке Паскаль, С++, С# или Java.

Тема 28. Создать простейший векторный графический редактор для рисования линий, различных геометрических фигур, заливки и т.д.

**Содержание**

[1. Формирование требований 4](#_Toc93845906)

[2. Проектирование 6](#_Toc93845907)

[3. Реализация 10](#_Toc93845908)

[3.1. Библиотека классов – классы фигур 10](#_Toc93845909)

[3.2. Библиотека классов – хранилище фигур 12](#_Toc93845910)

[3.3. Приложение – рисунок и рабочая область 16](#_Toc93845911)

[3.4. Приложение – инструменты рисования 19](#_Toc93845912)

[4. Список литературы 20](#_Toc93845913)

# 1. Формирование требований

Требуется разработать простейший векторный графический редактор, используя объектно-ориентированный подход. Определим следующие требования к функционалу приложения:

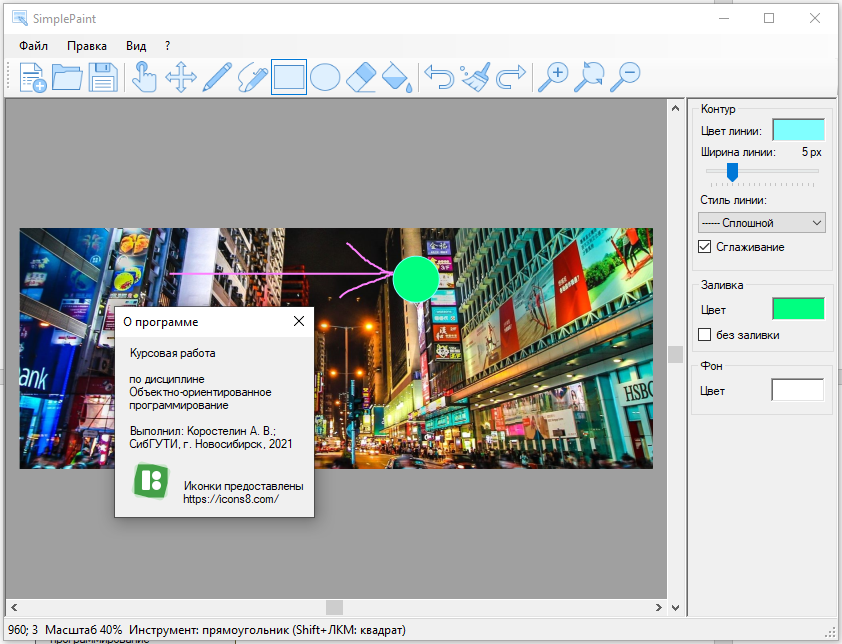
1. Создание рабочей области (рисунка) заданных размеров; импорт рисунка из растра JPG/PNG/BMP, экспорт рабочей области со всем её содержимым в растр JPG/PNG/BMP.
2. Наличие трёх инструментов рисования графических примитивов: прямых линий, прямоугольников, эллипсов.
3. Обработка модификатора Shift во время рисования. С зажатым Shift упомянутые выше инструменты рисуют линии под углом 45 и 90 градусов, квадраты и окружности соответственно.
4. Инструмент рисования произвольных кривых «от руки» и «ластик» для стирания произвольных частей изображения.
5. Инструмент заливки нарисованных фигур заданным цветом.
6. Инструмент перемещения уже нарисованных фигур в границах рабочей области.
7. Создаваемые пользователем графические объекты хранятся в перечислении, имеющем методы отмены и повторения изменений (механизм undo/redo).
8. Рабочая область может масштабироваться, а также перемещаться в границах окна приложения с помощью ползунков прокрутки или перетаскивания мышью.
9. Окно приложения имеет панель инструментов, панель выбора графических свойств (цвета, контуры и т.п.), строку состояния с текущим положением курсора, масштабом, подсказкой по выбранному инструменту.

Программа рассчитана на широкий круг пользователей, обладающих стандартными навыками работы с ПК. Каких-либо режимов разделения прав доступа, средств надзора, диагностических и административных инструментов не предусматривается.

Программа должна выполняется на IBM PC-совместимом компьютере под управлением ОС Microsoft Windows версий, актуальных на момент составления настоящих требований, разрядность x32/x64.

Программа поставляется в скомпилированном виде (файлы EXE, DLL), готовом для исполнения в целевой рабочей среде .NET Framework 4.7.2. Инструменты установки и удаления не предусматриваются.

Внешний вид окна приложения изображён на рисунке 1.1.



**Рис. 1.1.** Результат работы программы.

Для написания приложения использована среда разработки Microsoft Visual Studio 2019; язык C#. Классы графических объектов и класс-контейнер, реализующий механизм undo/redo, выделены в отдельную библиотеку.

К настоящей работе приложены:

1. исполняемый файл *SimplePaint.exe*;
2. библиотека классов *ShapesLibrary.dll*;
3. проект, содержащий весь исходный код (каталог *src*).

# 2. Проектирование

Определим внутреннюю логику работы приложения, используемые классы и вытекающие из этой логики взаимосвязи между ими.

Рисунок как некоторая сущность, имеющая геометрические размеры, прямоугольную форму, содержащая в себе некоторые геометрические фигуры и отображаемая на поверхности GDI+, описывается отдельным классом (class Drawing : IDrawing). Набор методов класса сформирован исходя из следующих соображений.

Фигуры, содержащиеся в рисунке, можно удалять, добавлять, и изменять – при этом данные действия можно отменить или повторить (т.е. реализован механизм undo/redo). В графических редакторах для изменения или удаления фигуры её прежде выбирают кликом мыши – соответственно и рисунок должен предоставлять метод, обеспечивающий данный функционал.

При любых изменениях данных, инкапсулированных в рисунке, если таковые изменения могут быть визуализированы, вызывается событие, которое может быть обработано приложением для отображения актуального содержимого рисунка на поверхности GDI+. Отображаемые фигуры сортируются по порядку на оси Z (те, которые «глубже», перекрываются теми, которые «выше»).

Общепринятой практикой является выдача пользователю запроса о сохранении данных при закрытии приложения, если им были внесены какие-либо изменения; в противном случае запрос не выдаётся. Для реализации этого функционала класс имеет свойство, сообщающее о наличии изменений.

Каждая фигура, существующая в рамках рисунка, является объектом соответствующего класса. Верхним в иерархии является абстрактный класс Shape, реализующий интерфейсы IDrawable. Также реализован системный интерфейс ICloneable для создания глубокой копии объекта. Потомками Shape являются классы Line, Rectngl, Ellipe и Freepath, представляющие собой линию, прямоугольник/квадрат, эллипс/окружность и произвольный контур соответственно.

Внутри рисунка (т.е. объекта IDrawing) инкапсулировано хранилище геометрических фигур (т.е. объектов IDrawable), представленное экземпляром класса class RewindableShapesStorage : IRewindableShapesStorage. Важной особенностью данного хранилища является то, что изменения его содержимого могут быть отменены, а отменённые – возвращены обратно. Механизм undo/redo рисунка фактически взаимодействует с соответствующими методами данного хранилища.

Классы фигур IDrawable и хранилища IRewindableShapesStorage вынесены в отдельную библиотеку классов ShapesLibrary.

Каждая фигура характеризуется комплексом визуализируемых свойств, таких как цвет и тип заливки, цвет и толщина контура, тип штрихов контура. В программе предусмотрена панель инструментов, позволяющая пользователю задавать эти свойства. Будучи заданными, они инкапсулируются в объект класса Palette.

Важнейшим элементом интерфейса графического редактора является рабочая область (она же область рисования, в некоторых программах называется «холст»), визуализирующая рисунок. В данной программе рабочая область является пользовательским компонентом, унаследованным от UserControl (class DrawCanvas : UserControl). Ключевой особенностью данного компонента является способность к масштабированию. Вызывая соответствующие методы, пользователь может увеличивать или уменьшать размер рабочей области, при этом визуализируемые графические объекты также изменяют свои размеры пропорционально.

Поскольку рабочая область может иметь произвольный масштаб, а масштаб объектов внутри рисунка всегда равен 100%, при работе с мышью возникает необходимость пересчёта координат курсора, а при отображении графических объектов – необходимость пересчёта их размеров с учётом коэффициента масштабирования. Объект DrawCanvas предоставляет три события мыши, event MouseEventHandler OnMouseDownScaled, OnMouseUpScaled и OnMouseMoveScaled, аналогичные стандартным MouseDown, MouseUp, MouseMove, но предоставляющие уже пересчитанные с учетом текущего масштаба аргументы MouseEventArgs. При каждом обновлении рабочей области вызывается событие event PaintEventHandler ShapesDrawRequest. Как и в случае с мышью, аргументы события PaintEventArgs содержат уже масштабированный объект Graphics, представляющий поверхность для рисования графических объектов. Таким образом, масштабирование заключено внутри компонента DrawCanvas, и в остальных частях программы нет необходимости учитывать это.

Программа предоставляет пользователю определённые инструменты для работы с графическими объектами (рисование фигур, заливка и т.д.). Каждый существующий в программе инструмент является объектом соответствующего класса. Верхним в иерархии является абстрактный класс DrawingTool, реализующий интерфейс IDrawingTool. От него унаследованы остальные классы инструментов, всего их 8, но благодаря объектному подходу количество может быть увеличено без существенных изменений в коде программы.

Для удобства, все доступные инструменты реализованы в виде набора, из которого пользователь выбирает нажатием соответствующей кнопки на панели тот инструмент, который ему требуется. Такой набор инкапсулирован в статическом классе (static class DrawToolBox). Выбор инструмента осуществляется вызовом статического метода Select(DrawingTools tool). Использование набора инструментов позволяет упростить код – все остальные части программы работают только с классом DrawToolBox, не учитывая и даже не зная, какие инструменты существуют, какой из них выбрал пользователь, и вообще выбирал ли он что-либо.

Также в программе определён вспомогательный статический класс static class PointMath, позволяющий складывать, вычитать, и масштабировать на заданный коэффициент точки (Point).

# 3. Реализация

Полный исходный код программы SimplePaint и библиотеки классов ShapesLibrary приложен к работе. Рассмотрим основные особенности классов.

## 3.1. Библиотека классов – классы фигур

Каждый графический примитив, существующий в рамках рисунка, является объектом соответствующего класса. Верхним в иерархии является абстрактный класс Shape, реализующий интерфейсы IDrawable и системный интерфейс ICloneable. Потомками его являются классы Line, Rectngl, Ellipe и Freepath, представляющие собой линию, прямоугольник/квадрат, эллипс/окружность и произвольный контур соответственно.

Конструктор каждого класса принимает не менее трёх необходимых аргументов, определяющих: контур объекта (тип Pen), заливку (тип Brush, может быть null для объектов без заливки), координаты расположения на рисунке (тип Point).

Интерфейс графического класса определяет необходимые методы:

public interface IDrawable : ICloneable

{

void SetFill(Brush fillBrush);

void SetPen(Pen outlinePen);

void AddPoint(Point pathPoint, bool snapToStraight);

void Draw(Graphics drawSurface);

Rectangle GetBoundingRectangle();

bool HitTest(Point point, bool includeInside);

void Move(Point offset);

int ZOrder { get; set; }

}

Методы SetFill и SetPen задают заливку и обводку фигуры соответственно;

метод AddPoint добавляет в контур объекта очередную точку при рисовании (если аргумент snapToStraight установлен в true, добавляемая точка будет изменена таким образом, чтоб обеспечить рисование «правильной» фигуры – квадрата, круга, или притягивание линии к нормали);

метод Draw производит рисование фигуры на заданной поверхности, используя методы класса Graphics;

метод GetBoundingRectangle возвращает прямоугольник, в который вписана данная фигура – может быть использован, чтоб не обновлять всю поверхность GDI при изменении одной фигуры, а только ту её часть, которая собственно фигуру отображает;

метод HitTest позволяет определить, находится ли данная точка на контуре фигуры (или внутри фигур, если includeInside установлен в true). Используется для определения, был ли клик мышью произведён в границах фигуры или нет, в инструментах заливки и перемещения фигур.

метод Move перемещает фигуру на заданную величину смещения.

Свойство ZOrder задаёт положение объекта на оси Z относительно остальных объектов (чем больше, тем фигура «глубже»).

Метод Clone, реализующий интерфейс ICloneable, возвращает глубокую копию объекта.

## 3.2. Библиотека классов – хранилище фигур

Для хранения коллекции объектов IDrawable, существующих в рамках рисунка, используется специальный класс, реализующий возможность отмены и повтора изменений, описывающийся следующим интерфейсом:

public interface IRewindableShapesStorage

{

int Count { get; } //число объектов в коллекции

void Add(IDrawable shape); //добавить объект в коллекцию

IEnumerable<IDrawable> GetSortedContents(); //получить все объекты, упорядоченные по оси Z

IDrawable ElementAt(int index); //получить объект по его индексу

void Replace(int index, IDrawable newshape);//заменить объект по индексу

void RemoveAt(int index); //удалить объект по индексу

void Rewind(); //шаг назад – отмена действия

void Forward(); //шаг вперёд – повтор отменённого

void RewindFull(); //возврат к исходному состоянию – отмена всех действий

}

Таким образом, все действия пользователя с графическими объектами сводятся к трём: добавление объекта (нарисована новая фигура), удаление объекта (удалена нарисованная ранее фигура), замена существующего объекта новым (изменены свойства существующей фигуры – заливка или расположение). Перечисление этих действий:

private enum DrawingOperations

{

Create,

Update,

Delete

}

Каждое действие может быть отменено (undo), а отменённые действия затем могут быть возвращены обратно (redo). При добавлении нового объекта, возврат отменённого становится невозможным, что является общепринятым поведением для этого механизма (в противном случае, возникала бы проблема двойственности – нельзя определить, в каком шаге действий вновь добавленная фигура должна существовать, а в каком - нет).

Для реализации механизма undo/redo внутри класса содержится две коллекции. Список private readonly List<IDrawable> shapes хранит текущий набор фигур. Другой список private readonly List<(IDrawable, int, DrawingOperations)> timeline хранит историю изменений этого набора. Каждый элемент списка – кортеж значений, характеризующих произошедшее изменение: (предшествующее состояние фигуры, индекс фигуры в списке, выполненное действие). Список timeline можно представить в виде временно́й ленты, каждый кадр которой – некое событие. Также класс содержит переменную private int currtime хранящую «текущее время» - т.е. индекс текущего кадра. Методы класса Rewind() и Forward() перематывают ленту на кадр назад или вперёд, декрементируя или инкрементируя значение этой переменной:

public void Rewind()

{

if (currtime <= 0)

{

return;

}

ExchangeWithSnapshot(--currtime);

}

public void Forward()

{

if (currtime >= timeline.Count)

{

return;

}

ExchangeWithSnapshot(currtime++);

}

public void RewindFull()

{

while (currtime > 0)

{

Rewind();

}

}

При каждой смене кадра вызывается метод private void ExchangeWithSnapshot(int timelineIndex), приводящий соответствующую фигуру в основном списке shapes в то состояние, которое сохранено в текущем кадре. При этом в содержимое кадра записывается текущее состояние фигуры (т.е. происходит обмен состояниями). Таким образом, при перемотке в противоположном направлении состояние фигуры восстановится, и так сколько угодно раз для любого количества фигур в списке.

Алгоритм работы метода ExchangeWithSnapshot следующий:

1. Из значений текущего кадра (элемент списка timeline с индексом timelineIndex) получаем тип текущей операции, индекс затрагиваемой ей фигуры в списке shapes и предыдущее значение этой фигуры.
2. Если текущая операция – создание фигуры, то в содержимое текущего кадра записываем текущую фигуру и тип операции – удаление, а из списка shapes фигуру удаляем.
3. Если текущая операция – удаление фигуры, то предыдущее значение фигуры из текущего кадра вставляем в список shapes по соответствующему индексу; в содержимое текущего кадра записываем тип операции – создание.
4. Если текущая операция – изменение фигуры, то меняем фигуры местами: та, что сохранена в кадре – в список shapes, а та, что была в списке – в текущий кадр.

Соответственно, реализация метода:

private void ExchangeWithSnapshot(int timelineIndex)

{

int currentIndex = timeline[timelineIndex].Item2;

IDrawable currentShape = timeline[timelineIndex].Item1;

DrawingOperations currentOperation = timeline[timelineIndex].Item3;

if (currentIndex < 0 || currentIndex > shapes.Count)

{

return;

}

switch (currentOperation)

{

case DrawingOperations.Create:

IDrawable shapetosave = shapes.ElementAt(currentIndex);

shapes.RemoveAt(currentIndex);

timeline[timelineIndex] = (shapetosave, currentIndex, DrawingOperations.Delete);

break;

case DrawingOperations.Update:

shapetosave = shapes[currentIndex];

shapes[currentIndex] = currentShape;

timeline[timelineIndex] = (shapetosave, currentIndex, DrawingOperations.Update);

break;

case DrawingOperations.Delete:

shapes.Insert(currentIndex, currentShape);

timeline[timelineIndex] = (null, currentIndex, DrawingOperations.Create);

break;

}

}

При внесении изменений в список shapes извне, т.е. вызове методов Add(), Replace(), RemoveAt() класса RewindableShapesStorage, происходит добавление кадра, хранящего предыдущее состояние затрагиваемой фигуры, в timeline по индексу, определяемому текущим временем, т.е. значением переменной currtime. После каждого значимого действия эта переменная инкрементируется.

public void Replace(int index, IDrawable newshape)

{

if (index < 0 || index >= shapes.Count || newshape is null)

{

return;

}

timeline.Insert(currtime++, (shapes[index], index, DrawingOperations.Update));

shapes[index] = newshape;

}

public void RemoveAt(int index)

{

if (index < 0 || index >= shapes.Count)

{

return;

}

timeline.Insert(currtime++, (shapes[index], index, DrawingOperations.Delete));

shapes.RemoveAt(index);

}

Также, при добавлении новой фигуры, стирается всё содержимое ленты событий, находящееся «в будущем» по отношению к текущему моменту времени:

public void Add(IDrawable shape)

{

if (shape is null)

{

return;

}

shapes.Add(shape);

timeline.Insert(currtime++, (null, shapes.Count - 1, DrawingOperations.Create));

timeline.RemoveRange(currtime, timeline.Count - currtime);

}

## 3.3. Приложение – рисунок и рабочая область

Рисунок как некоторая сущность, имеющая геометрические размеры и содержащая набор графических объектов, описан классом, реализующим интерфейс IDrawing:

public delegate void UpdateHandler(Rectangle updatedBounds);

internal interface IDrawing

{

event UpdateHandler Updated; //событие возникает при изменении содержимого

bool DrawingChanged { get; } //true если рисунок изменялся после создания

Size Size { get; set; } //размеры рисунка

void AddShape(IDrawable shape); //добавить фигуру к рисунку

void AddBitmap(Bitmap bitmap, bool resizeDrawing); //задать растр как фоновое изображение, если resizeDrawing – подогнать размер рисунка по размеру растра

void DrawAll(Graphics drawSurface); //вывести все обьекты на поверхность drawSurface, соблюдая порядок по оси Z

void Undo(); //отменить действие

void Redo(); //повторить действие

void UndoAll(); //отменить все действия

IDrawable SelectShapeByPoint(Point ptToSearch, bool includeInside = false);

//пометить графический обьект как выбранный и вернуть ссылку на него, если предоставленная

//точка включена в контур объекта (или находится внутри фигуры, если includeInside);

//в противном случае вернуть null и снять выбор с обьекта, выбранного ранее

void MoveSelectedShape(Point offset); //переместить выбранный обьект

void FillSelectedShape(Brush fill); //заливка выбранного объекта

void DiscardSelectedShape(); //удаление выбранного обьекта

}

При каждом визуализируемом изменении содержимого рисунка вызывается событие Updated.

Рисунок содержит счетчик объектов, инкрементирующийся при каждом добавлении нового. Значение счётчика присваивается полю ZOrder добавляемой фигуры, таким образом, каждая вновь добавленная фигура отображается поверх предыдущих.

Рисунок содержит хранилище объектов RewindableShapesStorage. Методы отмены и повтора действия, добавления, изменения и удаления фигур взаимодействуют с соответствующими методами хранилища.

Поле DrawingChanged будет истинно, если хранилище не пустое или был присвоен фоновый растр.

С понятием рисунка тесно связано понятие рабочей области. Но если рисунок (класс Drawing) является абстракцией, инкапсулирующей графические объекты и предоставляющей методы взаимодействия с ними, то рабочая область (компонент DrawCanvas) – это элемент интерфейса, отображающий рисунок на экране и вызывающий события, когда пользователь взаимодействует с рисунком посредством мыши. Кроме того, как упомянуто в предыдущем разделе, это компонент скрывает внутри себя работу с масштабированием изображения.

Реализация класса проста и не требует особых пояснений:

public partial class DrawCanvas : UserControl

{

private const float DEFAULT\_ZOOM\_FACTOR = 1.0F; //масштаб по умолчанию 100%

private const float ZOOM\_STEP = 0.1F; //шаг изменения масштаба 10%

public float ZoomFactor { get; set; } = DEFAULT\_ZOOM\_FACTOR; //текущий масштаб

public Size SizeOriginal { get; private set; } //размер при масштабе 100%

public SmoothingMode Smoothing { get; set; } = SmoothingMode.None; //сглаживание

public event PaintEventHandler ShapesDrawRequest;

//вызывается при отрисовке компонента; может быть использовано для отрисовки фигур

//переоперелённые события мыши предоставляют e.Location пересчитанные с учетом масштаба

public event MouseEventHandler OnMouseDownScaled;

public event MouseEventHandler OnMouseUpScaled;

public event MouseEventHandler OnMouseMoveScaled;

public DrawCanvas()

{

InitializeComponent();

DoubleBuffered = true;

SizeOriginal = Size;

}

public void SetSize(Size size)

{

SizeOriginal = size;

this.Size = size;

}

public void ZoomIn()

{

ZoomFactor += ZOOM\_STEP;

Invalidate();

}

public void ZoomOut()

{

if (ZoomFactor <= ZOOM\_STEP)

{

return;

}

ZoomFactor -= ZOOM\_STEP;

Invalidate();

}

public void ZoomReset()

{

ZoomFactor = 1.0F;

Invalidate();

}

public Graphics GetGraphics() //возвращает масштабированный this.Graphics

{

Graphics gr = CreateGraphics();

gr.ScaleTransform(ZoomFactor, ZoomFactor);

return gr;

}

public Bitmap GetBitmap() //отрисовывает содержимое в растр

{

ZoomReset();

Bitmap btmp = new Bitmap(Width, Height, CreateGraphics());

DrawToBitmap(btmp, new Rectangle(0, 0, Width, Height));

return btmp;

}

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

e.Graphics.SmoothingMode = Smoothing;

e.Graphics.ScaleTransform(ZoomFactor, ZoomFactor);

ShapesDrawRequest?.Invoke(this, e);

Size = new Size((int)(SizeOriginal.Width \* ZoomFactor), (int)(SizeOriginal.Height \* ZoomFactor));

}

protected override void OnMouseDown(MouseEventArgs e)

{

base.OnMouseDown(e);

Point unscaledPt = PointMath.UnscalePoint(e.Location, ZoomFactor);

OnMouseDownScaled?.Invoke(this, new MouseEventArgs(e.Button, e.Clicks, unscaledPt.X, unscaledPt.Y, e.Delta));

}

protected override void OnMouseUp(MouseEventArgs e)

{

base.OnMouseUp(e);

Point unscaledPt = PointMath.UnscalePoint(e.Location, ZoomFactor);

OnMouseUpScaled?.Invoke(this, new MouseEventArgs(e.Button, e.Clicks, unscaledPt.X, unscaledPt.Y, e.Delta));

}

protected override void OnMouseMove(MouseEventArgs e)

{

base.OnMouseMove(e);

Point unscaledPt = PointMath.UnscalePoint(e.Location, ZoomFactor);

OnMouseMoveScaled?.Invoke(this, new MouseEventArgs(e.Button, e.Clicks, unscaledPt.X, unscaledPt.Y, e.Delta));

}

}

## 3.4. Приложение – инструменты рисования

Программа предоставляет пользователю определённые инструменты для работы с графическими объектами (рисование фигур, заливка и т.д.). Каждый существующий в программе инструмент является объектом соответствующего класса. Верхним в иерархии является абстрактный класс DrawingTool, реализующий интерфейс IDrawingTool. От него унаследованы остальные классы инструментов, всего их 8, но благодаря объектному подходу количество может быть увеличено без существенных изменений в коде программы. Перечисление существующих инструментов:

internal enum DrawingTools

{

None,

Selector, //выбор мышью фигуры на рисунке и её перемещение

Move, //перемещение мышью рабочей области в окне программы

Pencil, //создание линий

Freehand, // создание произвольных контуров

Rectangle, // создание прямугольников (квадратов)

Ellipse, // создание эллипсов (окружностей)

Eraser, //ластик

Fill //заливка фигур

}

Поскольку операции с графическими объектами выполняются посредством мыши, то для реализации любого инструмента минимально необходимо и достаточно обрабатывать события мыши, возникающие в компоненте рабочей области (т.е. объекте DrawCanvas). Фактически, инструмент обрабатывает события мыши и производит некие изменения рисунка и/или рабочей области, с учетом заданных пользователем визуализируемых свойств объектов. Поэтому, конструктор класса инструмента в качестве аргументов требует ссылки на текущий рисунок (объект IDrawing), текущую рабочую область (объект DrawCanvas), текущий набор визуализируемых свойств (объект Palette), а интерфейс класса выглядит следующим образом:

internal interface IDrawingTool

{

void ProcessMouseDown(MouseEventArgs e);

void ProcessMouseMove(MouseEventArgs e);

void ProcessMouseUp(MouseEventArgs e);

}

Для удобного изменения инструментов и добавления новых, приложение не создаёт объекты классов инструментов и не взаимодействует с ними напрямую. Вместо этого, работа с инструментами осуществляется через статический класс DrawToolBox, представляющий собой набор доступных инструментов. Использование набора инструментов позволяет упростить код – все остальные части программы работают только с классом DrawToolBox, не учитывая и даже не зная, какие инструменты существуют, какой из них выбрал пользователь, и вообще выбирал ли он что-либо.

При нажатии на кнопку на панели инструментов, происходит выбор соответствующего инструмента в наборе вызовом статического метода Select(DrawingTools tool). События движения мыши, порождаемые в компоненте DrawCanvas, также передаются для обработки в набор инструментов, где передаются выбранному инструменту.

internal static class DrawToolBox

{

private static IDrawingTool currentTool;

private static Palette currentPalette;

private static IDrawing currentDrawing;

private static DrawCanvas currentCanvas;

public static Palette CurrentPalette

{ get => currentPalette; set => currentPalette = value; }

static DrawToolBox()

{

currentPalette = new Palette();

}

public static void SetDrawing(IDrawing drawing)

{

currentDrawing = drawing;

}

public static void SetCanvas(DrawCanvas canvas)

{

currentCanvas = canvas;

}

public static void Select(DrawingTools tool)

{

switch (tool)

{

case DrawingTools.None:

currentTool = null;

break;

case DrawingTools.Pencil:

currentTool = new GenericTool<Line>(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Freehand:

currentTool = new GenericTool<Freepath>(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Rectangle:

currentTool = new GenericTool<Rectngl>(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Ellipse:

currentTool = new GenericTool<Ellipse>(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Eraser:

currentTool = new ToolEraser(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Move:

currentTool = new ToolCanvasMove(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Selector:

currentTool = new ToolShapeSelect(currentPalette, currentCanvas,

currentDrawing);

break;

case DrawingTools.Fill:

currentTool = new ToolShapeFill(currentPalette, currentCanvas,

currentDrawing);

break;

}

}

public static void ProcessMouseDown(MouseEventArgs e)

{

currentTool?.ProcessMouseDown(e);

}

public static void ProcessMouseMove(MouseEventArgs e)

{

currentTool?.ProcessMouseMove(e);

}

public static void ProcessMouseUp(MouseEventArgs e)

{

currentTool?.ProcessMouseUp(e);

}

}

Поскольку каждая фигура описывается отдельным классом, несложно заметить, что инструменты Pencil, Freehand, Rectangle, Ellipse фактически выполняют идентичное действие, а именно создают обьект IDrawable; различается лишь тип создаваемого объекта. Поэтому все они реализованы одним и тем же обобщённым классом:

class GenericTool<T> : DrawingTool where T : IDrawable

Этот обобщённый класс взаимодействует с «фабрикой объектов» - статическим классом (static class ShapesFactory), создающим объект IDrawable типа T. Рассмотрим данный класс:

internal partial class GenericTool<T> : DrawingTool where T : IDrawable

{

public GenericTool(Palette palette, DrawCanvas canvas, IDrawing drawing) : base(palette, canvas, drawing) { }

public override void ProcessMouseDown(MouseEventArgs e)

{

if (e.Button != MouseButtons.Left)

{

return;

}

if (canvas is null)

{

throw new NullReferenceException();

}

Cursor.Current = Cursors.Cross;

Cursor.Clip = new Rectangle(canvas.PointToScreen(Point.Empty), canvas.Size);

ShapesFactory.Init<T>(palette.ForegroundPen, palette.FillBrush, e.Location);

}

public override void ProcessMouseMove(MouseEventArgs e)

{

if (canvas is null)

{

throw new NullReferenceException();

}

if (e.Button != MouseButtons.Left)

{

return;

}

canvas.Refresh();

ShapesFactory.AddPoint(e.Location, Control.ModifierKeys == Keys.Shift);

ShapesFactory.Finish().Draw(canvas.GetGraphics());

}

public override void ProcessMouseUp(MouseEventArgs e)

{

if (drawing is null)

{

throw new NullReferenceException();

}

if (e.Button != MouseButtons.Left)

{

return;

}

drawing?.AddShape(ShapesFactory.Finish());

Cursor.Current = Cursors.Arrow;

Cursor.Clip = Rectangle.Empty;

}

}

Как видно из кода, при нажатии левой кнопки мыши происходит инициализация фабрики объектов, при перемещении курсора с нажатой левой кнопкой передаются текущие координаты, и при отпускании кнопки переданная из фабрики объектов фигура добавляется в содержимое текущего рисунка.

Сама же фабрика объектов фактически создаёт объект заданного типа, взаимодействуя с конструктором, передаёт координаты соответствующему методу и возвращает созданный объект.

# 4. Список литературы

1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б.— Электрон. текстовые данные.— М.: Интернет- Университет Информационных Технологий (ИНТУИТ), 2016.— 285 c. - Режим доступа: http://www.iprbookshop.ru/39552.html по паролю – ЭБС «IPRbooks».

2. Комлев Н.Ю. Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей [Электронный ресурс]/ Комлев Н.Ю.— Электрон. текстовые данные.— М.: СОЛОН-ПРЕСС, 2014.— 298 c. - Режим доступа: http://www.iprbookshop.ru/26923.html по паролю – ЭБС «IPRbooks».

3. Шилдт Г. C# 4.0: полное руководство.— М.: Вильямс, 2012.— 1056 c.

4. Макконнелл С. Совершенный код. Мастер-класс. — СПб.: БХВ, 2021.— 896 с.

5. Microsoft .NET Framework API Reference version 4.7.2 [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/dotnet/api/?view=netframework-4.7.2