**Міністерство освіти і науки України Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки Кафедра обчислювальної техніки**

**Лабораторна робота 5.**

з дисципліни

«Об’єктно орієнтоване програмування» на тему

**Розробка багатовіконного інтерфейсу**

**користувача для графічного редактора об’єктів**

Виконав: Перевірив:

Студент групи ІМ-22 Порєв В.М.

Куц Іван Васильович

номер у списку групи: 12

Київ 2023

# Мета

Отримати вміння та навички програмувим багатовіконний інтерфейс програми на С++ в об’єктно-орієнтованому стилі.

# Завдання

1. Створити у середовищі MS Visual Studio C++ проект Desktop Application з ім’ям **Lab5**.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Перевірити роботу програми. Налагодити програму.
4. Проаналізувати та прокоментувати результати та вихідний текст програми
5. Оформити звіт

**Вимоги згідно з варіантом :**

1. Реалізація Singltone - класична

2. Такі ж кольори і стилі як у попередній лабораторній

3. Колір заливки — як у попередній роботі

4. Запрограмувати вікно таблиці, для відкриття якого передбачити окремий пункт у меню. При закритті головного вікна передбачити закриття вікна таблиці

5. Таблиця повинна бути запрограмована як окремий модуль, саме ж вікно - немодальне

6. Передбачити запис у файл об’єктів шо вводяться

7. Архітекрута повинна бути зручною для додавання нових типів об’єктів

8. Зроби діаграму класів

9. Додатково можна запрограувати наступний функіонал:

1.1. Видалення з таблиці об’єктів

# Вихідний текст програмного коду

## MyEditor.ts

import { IpcRendererEvent } from 'electron'  
import Shape from './shapes/Shape'  
import Line from './shapes/Line'  
import Point from './shapes/Point'  
import Ellipsis from './shapes/Ellipsis'  
import Rectangle from './shapes/Rectangle'  
import Cube from './shapes/Cube'  
import LineOO from './shapes/LineOO'  
// eslint-disable-next-line @typescript-eslint/ban-ts-comment  
// @ts-ignore  
class MyEditor {  
 private canvas: HTMLCanvasElement  
 private ctx: CanvasRenderingContext2D  
 private toolbarButtons!: HTMLInputElement[]  
 private currentShape: Shape  
 private isPainting: boolean = false  
 private fillColor: string = '#000000'  
 private outlineColor: string = '#64ff00'  
  
 constructor(public shapes: Shape[] = []) {  
 this.canvas = document.getElementById('canvas') as HTMLCanvasElement  
 this.ctx = this.canvas.getContext('2d') as CanvasRenderingContext2D  
 this.currentShape = new Rectangle(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
  
 this.canvas.addEventListener('mousedown', (event) => {  
 this.startShapePaint(event)  
 })  
  
 this.canvas.addEventListener('mousemove', (e) => {  
 this.paintShape(e)  
 })  
 this.canvas.addEventListener('mouseup', (e) => {  
 this.endShapePaint(e)  
 })  
 this.configureToolbar()  
 this.configureAdditionalTools()  
  
 window.electron.ipcRenderer.on('delete-shape-event', this.hadleDeleteShapeEvent.bind(this))  
 }  
  
 private startShapePaint(event: MouseEvent) {  
 this.isPainting = true  
 this.currentShape = this.currentShape?.getInstance(  
 event,  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 }  
 private paintShape(\_: MouseEvent) {  
 if (!this.isPainting) return  
 this.canvas.classList.add('painting')  
 this.rerenderCanvas()  
 this.currentShape?.changePosition(\_)  
 this.currentShape?.paintOutline(this.ctx)  
 }  
 private endShapePaint(\_: MouseEvent) {  
 if (this.currentShape === undefined) return  
 this.isPainting = false  
 this.shapes.push(this.currentShape as Shape)  
 this.canvas.classList.remove('painting')  
 this.rerenderCanvas()  
 }  
  
 private rerenderCanvas() {  
 this.repaintShapes()  
 this.emitRenderShapesEvent()  
 this.writeShapesToFile()  
 }  
  
 private repaintShapes() {  
 this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height)  
 this.shapes.forEach((shape) => shape.paint(this.ctx))  
 }  
 private async writeShapesToFile() {  
 window.api.writeFileSync('shapes.json', JSON.stringify(this.shapes))  
 }  
 private emitRenderShapesEvent() {  
 window.electron.ipcRenderer.send('render-shapes-event', this.shapes)  
 }  
 private configureToolbar() {  
 this.toolbarButtons = [  
 '#ellipsis-btn',  
 '#line-btn',  
 '#rectangle-btn',  
 '#point-btn',  
 '#cube-btn',  
 '#lineOO-btn'  
 ].map((btn) => document.querySelector(btn) as HTMLInputElement)  
  
 this.toolbarButtons[0].addEventListener('click', () => {  
 this.currentShape = new Ellipsis(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 this.toolbarButtons[1].addEventListener('click', () => {  
 this.currentShape = new Line(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 this.toolbarButtons[2].addEventListener('click', () => {  
 this.currentShape = new Rectangle(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 this.toolbarButtons[3].addEventListener('click', () => {  
 this.currentShape = new Point(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 this.toolbarButtons[4].addEventListener('click', () => {  
 this.currentShape = new Cube(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 this.toolbarButtons[5].addEventListener('click', () => {  
 this.currentShape = new LineOO(  
 new MouseEvent('click'),  
 this.ctx,  
 this.fillColor,  
 this.outlineColor  
 )  
 })  
  
 for (const button of this.toolbarButtons) {  
 button.addEventListener('click', (event) => {  
 const targetedButton = event.target as HTMLInputElement  
 this.toolbarButtons.forEach((b) => b.classList.remove('selected'))  
 targetedButton.classList.add('selected')  
 })  
 }  
 }  
 private configureAdditionalTools() {  
 const cleanButton = document.querySelector('#clean-btn') as HTMLButtonElement  
 const backButton = document.querySelector('#back-btn') as HTMLButtonElement  
 const fillColorInput = document.querySelector('#fill-color') as HTMLInputElement  
 const outlineColorInput = document.querySelector('#outline-color') as HTMLInputElement  
 const toggleTableButton = document.querySelector('#toggle-table-btn') as HTMLInputElement  
  
 cleanButton.addEventListener('click', () => {  
 this.shapes = []  
 this.repaintShapes()  
 })  
  
 backButton.addEventListener('click', () => {  
 this.shapes.pop()  
 this.repaintShapes()  
 })  
  
 fillColorInput.addEventListener('change', () => {  
 this.fillColor = fillColorInput.value  
 })  
  
 outlineColorInput.addEventListener('change', () => {  
 this.outlineColor = outlineColorInput.value  
 })  
  
 toggleTableButton.addEventListener('click', () => {  
 window.electron.ipcRenderer.send('show-table-window', this.shapes)  
 })  
 }  
  
 private hadleDeleteShapeEvent(\_: IpcRendererEvent, shapeId: number) {  
 this.shapes = this.shapes.filter((shape) => shape.id !== shapeId)  
 this.rerenderCanvas()  
 }  
}  
export default new MyEditor()

## Shape.ts

export default abstract class Shape {  
 static id: number = 0  
 public x0: number  
 public y0: number  
 public offsetX: number  
 public offsetY: number  
 public toolbarTitle: string  
 public initialEvent: MouseEvent  
 public fillColor: string  
  
 public id: number  
  
 protected constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 protected fillColor: string,  
 public outlineColor: string,  
 public toolbarTitle: string  
 ) {  
 this.initialEvent = event  
 this.offsetX = ctx.canvas.offsetLeft  
 this.offsetY = ctx.canvas.offsetTop  
 this.x0 = event.x - this.offsetX  
 this.y0 = event.y - this.offsetY  
 this.id = Shape.id  
 Shape.id++  
 }  
 abstract getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ): Shape  
 abstract changePosition(event: MouseEvent): void  
 abstract paintOutline(ctx: CanvasRenderingContext2D): void  
 abstract paint(ctx: CanvasRenderingContext2D): void  
}

## Rectangle.ts

import Shape from './Shape.d'  
export default class Rectangle extends Shape {  
 public x: number  
 public y: number  
  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Прямокутник')  
 this.x = this.x0  
 this.y = this.y0  
 }  
 changePosition(event: MouseEvent) {  
 this.x = event.clientX - this.offsetX  
 this.y = event.clientY - this.offsetY  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
  
 ctx.strokeRect(this.x0, this.y0, this.x - this.x0, this.y - this.y0)  
 ctx.fillRect(this.x0, this.y0, this.x - this.x0, this.y - this.y0)  
 }  
  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 ctx.setLineDash([5, 5])  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
  
 ctx.strokeRect(this.x0, this.y0, this.x - this.x0, this.y - this.y0)  
 ctx.setLineDash([0, 0])  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new Rectangle(event, ctx, fillColor, outlineColor)  
 }  
}

## Ellipsis.ts

import Shape from './Shape.d'  
export default class Ellipsis extends Shape {  
 public radiusX: number  
 public radiusY: number  
  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Еліпс')  
 this.radiusX = 0  
 this.radiusY = 0  
 }  
  
 changePosition(event: MouseEvent) {  
 this.radiusX = Math.abs(event.clientX - this.offsetX - this.x0)  
 this.radiusY = Math.abs(event.clientY - this.offsetY - this.y0)  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
  
 ctx.beginPath()  
 ctx.ellipse(this.x0, this.y0, this.radiusX, this.radiusY, 0, 0, 2 \* Math.PI)  
 ctx.fill()  
 ctx.stroke()  
 }  
  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
 ctx.setLineDash([5, 5])  
 ctx.beginPath()  
 ctx.ellipse(this.x0, this.y0, this.radiusX, this.radiusY, 0, 0, 2 \* Math.PI)  
 ctx.stroke()  
 ctx.setLineDash([0])  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new Ellipsis(event, ctx, fillColor, outlineColor)  
 }  
}

## Line.ts

import Shape from './Shape.d'  
export default class Line extends Shape {  
 public x: number  
 public y: number  
  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Лінія')  
 this.x = this.x0  
 this.y = this.y0  
 }  
 changePosition(event: MouseEvent) {  
 this.x = event.clientX - this.offsetX  
 this.y = event.clientY - this.offsetY  
 }  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 ctx.setLineDash([5, 5])  
  
 ctx.beginPath()  
  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
  
 ctx.moveTo(this.x0, this.y0)  
 ctx.lineTo(this.x, this.y)  
 ctx.stroke()  
 ctx.setLineDash([0])  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 ctx.beginPath()  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
  
 ctx.moveTo(this.x0, this.y0)  
 ctx.lineTo(this.x, this.y)  
 ctx.stroke()  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new Line(event, ctx, fillColor, outlineColor)  
 }  
}

# Cube.ts

import Shape from './Shape.d'  
import Rectangle from './Rectangle'  
import Line from './Line'  
  
export default class Cube extends Shape {  
 private frontRectangle: Rectangle  
 private backRectangle: Rectangle  
 private topLeftLine: Line  
 private topRightLine: Line  
 private bottomLeftLine: Line  
 private bottomRightLine: Line  
  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Куб')  
 this.frontRectangle = new Rectangle(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 this.backRectangle = new Rectangle(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 this.topLeftLine = new Line(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 this.topRightLine = new Line(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 this.bottomLeftLine = new Line(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 this.bottomRightLine = new Line(event, ctx, 'rgba(0,0,0,0)', outlineColor)  
 }  
  
 changePosition(event: MouseEvent) {  
 const newX = event.clientX - this.offsetX  
 const newY = event.clientY - this.offsetY  
  
 const dx = newX - this.x0  
  
 const cubeHeight = newY - this.y0  
 const dy = cubeHeight / 4  
  
 this.frontRectangle.x0 = this.backRectangle.x0 - dx  
 this.frontRectangle.x = newX  
 this.backRectangle.x = newX + dx  
  
 this.frontRectangle.y0 = this.backRectangle.y0 + dy  
 this.frontRectangle.y = newY  
 this.backRectangle.y = newY - dy  
  
 this.topLeftLine.x = this.frontRectangle.x0  
 this.topLeftLine.y = this.frontRectangle.y0  
  
 this.topRightLine.x0 = this.backRectangle.x0 + dx \* 2  
 this.topRightLine.y0 = this.backRectangle.y0  
  
 this.topRightLine.x = this.frontRectangle.x  
 this.topRightLine.y = this.frontRectangle.y - 3 \* dy  
  
 this.bottomLeftLine.y0 = this.backRectangle.y0 + dy \* 3  
 this.bottomLeftLine.y = this.frontRectangle.y0 + dy \* 3  
 this.bottomLeftLine.x = this.frontRectangle.x0  
  
 this.bottomRightLine.x0 = this.backRectangle.x  
 this.bottomRightLine.x = this.frontRectangle.x  
 this.bottomRightLine.y0 = this.backRectangle.y  
 this.bottomRightLine.y = this.frontRectangle.y  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 this.frontRectangle.paint(ctx)  
 this.backRectangle.paint(ctx)  
 this.topLeftLine.paint(ctx)  
 this.topRightLine.paint(ctx)  
 this.bottomLeftLine.paint(ctx)  
 this.bottomRightLine.paint(ctx)  
 }  
  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 ctx.setLineDash([5, 5])  
 this.frontRectangle.paintOutline(ctx)  
 this.backRectangle.paintOutline(ctx)  
 this.topLeftLine.paintOutline(ctx)  
 this.topRightLine.paintOutline(ctx)  
 this.bottomLeftLine.paintOutline(ctx)  
 this.bottomRightLine.paintOutline(ctx)  
 ctx.setLineDash([0])  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new Cube(event, ctx, fillColor, outlineColor)  
 }  
}

# LineOO.ts

import Shape from './Shape.d'  
import Ellipsis from './Ellipsis'  
import Line from './Line'  
export default class LineOO extends Shape {  
 protected leftEllipsis: Ellipsis  
 protected rightEllipsis: Ellipsis  
 protected line: Line  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Гантеля')  
 this.leftEllipsis = new Ellipsis(event, ctx, fillColor, outlineColor)  
 this.rightEllipsis = new Ellipsis(event, ctx, fillColor, outlineColor)  
 this.leftEllipsis.radiusX = 10  
 this.leftEllipsis.radiusY = 10  
 this.rightEllipsis.radiusX = 10  
 this.rightEllipsis.radiusY = 10  
 this.line = new Line(event, ctx, fillColor, outlineColor)  
 }  
  
 changePosition(event: MouseEvent) {  
 this.line.changePosition(event)  
 this.leftEllipsis.x0 = event.clientX - this.offsetX  
 this.leftEllipsis.y0 = event.clientY - this.offsetY  
 }  
  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 this.line.paintOutline(ctx)  
 this.leftEllipsis.paintOutline(ctx)  
 this.rightEllipsis.paintOutline(ctx)  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 this.line.paint(ctx)  
 this.leftEllipsis.paint(ctx)  
 this.rightEllipsis.paint(ctx)  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new LineOO(event, ctx, fillColor, outlineColor)  
 }  
}

# Point.ts

import Shape from './Shape.d'  
export default class Point extends Shape {  
 constructor(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 super(event, ctx, fillColor, outlineColor, 'Точка')  
 }  
 changePosition(event: MouseEvent) {  
 this.x0 = event.clientX - this.offsetX  
 this.y0 = event.clientY - this.offsetY  
 }  
  
 paintOutline(ctx: CanvasRenderingContext2D) {  
 this.paint(ctx)  
 }  
  
 paint(ctx: CanvasRenderingContext2D) {  
 ctx.fillStyle = this.fillColor  
 ctx.strokeStyle = this.outlineColor  
 ctx.fillRect(this.x0, this.y0, 3, 3)  
 }  
  
 getInstance(  
 event: MouseEvent,  
 ctx: CanvasRenderingContext2D,  
 fillColor: string,  
 outlineColor: string  
 ) {  
 return new Point(event, ctx, fillColor, outlineColor)  
 }  
}

## MyTable.ts

import Shape from './shapes/Shape'  
import type { IpcRendererEvent } from 'electron'  
class MyTable {  
 private tableBody: HTMLTableElement  
  
 constructor() {  
 this.tableBody = document.querySelector('.table-body') as HTMLTableElement  
 window.electron.ipcRenderer.on('render-shapes-table', this.renderShapes.bind(this))  
 }  
  
 renderShapes(\_: IpcRendererEvent, shapes: Shape[]) {  
 this.tableBody.innerHTML = ''  
 shapes.forEach((shape, i) => {  
 this.tableBody.innerHTML += `<tr data-id='${shape.id}'>  
 <td>${i + 1}</td>  
 <td>${shape.toolbarTitle}</td>  
 <td>(${shape.x0}, ${shape.y0})</td>  
 <td style='background-color:${shape.fillColor}'>${shape.fillColor}</td>  
 <td style='background-color:${shape.outlineColor}'>${shape.outlineColor}</td>  
 <td><button class='delete-btn' data-id='${shape.id}'></button></td>  
 </tr>`  
 })  
  
 const rows = this.tableBody.querySelectorAll('tr')  
 for (const row of rows) {  
 if (!row.dataset.id) continue  
 const shapeId = +row.dataset.id  
 const deleteBtn = row.querySelector('.delete-btn') as HTMLButtonElement  
 deleteBtn.addEventListener('click', () => {  
 this.emitDeleteShapeEvent(shapeId)  
 })  
 }  
 }  
  
 emitDeleteShapeEvent(id: number) {  
 window.electron.ipcRenderer.send('delete-shape-event', id)  
 }  
}  
  
export default new MyTable()

# Window.ts

import { BrowserWindow, BrowserWindowConstructorOptions } from 'electron'  
import { join } from 'path'  
import { is } from '@electron-toolkit/utils'  
  
const defaultProps: BrowserWindowConstructorOptions = {  
 width: 900,  
 height: 670,  
 show: false,  
 autoHideMenuBar: true,  
 webPreferences: {  
 devTools: true,  
 sandbox: false,  
 preload: join(\_\_dirname, '../preload/index.js')  
 }  
}  
  
export interface WindowConstructorParams extends BrowserWindowConstructorOptions {  
 file: string  
}  
class Window extends BrowserWindow {  
 constructor({ file, ...windowSettings }: WindowConstructorParams) {  
 super({ ...defaultProps, ...windowSettings })  
  
 this.once('ready-to-show', () => {  
 this.show()  
 })  
 const fileNameWithoutDir = file.split('/').at(-1)  
  
 if (is.dev && process.env['ELECTRON\_RENDERER\_URL']) {  
 this.loadURL(process.env['ELECTRON\_RENDERER\_URL'] + '/' + fileNameWithoutDir)  
 } else {  
 this.loadFile(file)  
 }  
 }  
}  
  
export default Window

# MyTableWindow.ts

import Window, { WindowConstructorParams } from './Window'  
  
class MyTableWindow extends Window {  
 private static windowInstance: MyTableWindow | null = null  
 private constructor(attributes: WindowConstructorParams) {  
 super(attributes)  
 this.on('close', (event) => {  
 event.preventDefault()  
 this.hide()  
 })  
 }  
 public getInstance(attributes?: WindowConstructorParams) {  
 if (!MyTableWindow.windowInstance) {  
 if (!attributes) throw new Error('Initialize an instance with attributes first')  
 MyTableWindow.windowInstance = new MyTableWindow(attributes)  
 }  
 return MyTableWindow.windowInstance  
 }  
}  
  
export default MyTableWindow

# Lab5.ts

import Window from './Window'  
import { join } from 'path'  
import MyTableWindow from './MyTableWindow'  
import { ipcMain } from 'electron'  
export default class Lab5 {  
 constructor() {  
 const mainWindow = new Window({  
 file: join(\_\_dirname, '../renderer/index.html')  
 })  
  
 let tableWindow: MyTableWindow  
  
 ipcMain.on('render-shapes-event', (\_, shapes) => {  
 if (!tableWindow) return  
 tableWindow.webContents.send('render-shapes-table', shapes)  
 })  
  
 ipcMain.on('delete-shape-event', (\_, shapeId) => {  
 if (!tableWindow) return  
 mainWindow.webContents.send('delete-shape-event', shapeId)  
 })  
  
 ipcMain.on('show-table-window', (\_, shapes) => {  
 tableWindow = MyTableWindow.prototype.getInstance({  
 file: join(\_\_dirname, '../renderer/table.html'),  
 parent: mainWindow  
 })  
 tableWindow.webContents.send('render-shapes-table', shapes)  
 tableWindow.show()  
 })  
 }  
}

# main.ts

iimport { app, ipcMain } from 'electron'  
import { electronApp, optimizer } from '@electron-toolkit/utils'  
import Lab5 from './Lab5'  
function main(): void {  
 const lab5 = new Lab5()  
  
 electronApp.setAppUserModelId('com.electron')  
  
 app.on('browser-window-created', (\_, window) => {  
 optimizer.watchWindowShortcuts(window)  
 })  
}  
  
app.whenReady().then(main)  
  
app.on('window-all-closed', () => {  
 app.quit()  
})

# styles.css

:root {  
 font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;  
 line-height: 1.5;  
 font-weight: 400;  
  
 color-scheme: light dark;  
 color: rgba(255, 255, 255, 0.87);  
 background-color: #3a3939;  
  
 font-synthesis: none;  
 text-rendering: optimizeLegibility;  
 -webkit-font-smoothing: antialiased;  
 -moz-osx-font-smoothing: grayscale;  
 -webkit-text-size-adjust: 100%;  
}  
  
  
body {  
 margin: 0;  
 display: flex;  
 flex-direction: column;  
 min-width: 320px;  
 padding: 20px 40px;  
 height: calc(100vh - 100px);  
}  
  
header{  
 display: flex;  
 justify-content: flex-start;  
}  
  
  
h1 {  
 font-size: 3.2em;  
 line-height: 1.1;  
}  
  
.logo {  
 height: 6em;  
 padding: 1.5em;  
 will-change: filter;  
 transition: filter 300ms;  
}  
.logo:hover {  
 filter: drop-shadow(0 0 2em #646cffaa);  
}  
.logo.vanilla:hover {  
 filter: drop-shadow(0 0 2em #3178c6aa);  
}  
  
.card {  
 padding: 2em;  
}  
  
.read-the-docs {  
 color: #888;  
}  
  
button {  
 border-radius: 8px;  
 border: 1px solid springgreen;  
 padding: 0.2em .6em;  
 font-size: 1em;  
 font-weight: 500;  
 font-family: inherit;  
 background-color: transparent;  
 cursor: pointer;  
 transition: border-color 0.25s;  
}  
button:hover {  
 border-color: #ddec0a;  
}  
  
  
  
@media (prefers-color-scheme: light) {  
 :root {  
 color: #213547;  
 background-color: #ffffff;  
 }  
 a:hover {  
 color: #747bff;  
 }  
 button {  
 background-color: #f9f9f9;  
 }  
}  
.navbar{  
 &>ul{  
 display: flex;  
 gap: 1rem;  
 padding-inline-start: 0;  
 &>li{  
 display: flex;  
 flex-direction: column;  
 align-items: flex-start;  
 }  
  
 }  
  
}  
  
ul{  
 list-style-type: none;  
 margin-block-start:0rem;  
}  
main{  
 min-height: 100%;  
 & canvas{  
 height: 500px;  
 width: 800px;  
 border: 2px solid grey;  
 border-radius: 6px;  
 transition: all ease-in-out .25s;  
  
 }  
 .painting{  
 border: 2px solid #b7b7b7;  
 cursor: crosshair;  
 }  
}  
  
  
#object\_mode{  
 font-weight: 600;  
}  
  
  
#toolbar{  
 padding: 0;  
 margin: 0;  
 display: flex;  
 gap: 1rem;  
 & input{  
 width: 34px;  
 height: 34px;  
 border: 2px solid transparent;  
 border-radius: 3px;  
 background-repeat: no-repeat;  
 cursor: pointer;  
 color: transparent;  
  
 }  
}  
  
#ellipsis-btn{  
 background-image: url("../assets/svg/ellipsis.svg");  
  
}  
#rectangle-btn{  
 background-image: url("../assets/svg/rectangle.svg");  
}  
#line-btn{  
 background-image: url("../assets/svg/line.svg");  
}  
#point-btn{  
 background-image: url("../assets/svg/point.svg");  
}  
#cube-btn{  
 background-image: url("../assets/svg/cube.svg");  
}  
#lineOO-btn{  
 background-image: url("../assets/svg/lineOO.svg");  
}  
#toggle-table-btn{  
 background-image: url("../assets/svg/table.svg");  
 width: 34px;  
 height: 34px;  
}  
  
  
.selected{  
 border: 2px solid springgreen !important;  
}  
  
.color-input{  
 width: 75%;  
 align-self: center;  
}  
  
  
#table{  
 border: 1px solid grey;  
 width:100%;  
 margin-bottom: 1rem;  
}  
  
#table thead{  
 position: sticky;  
 top: 0;  
 background: #3a3939;  
 border-bottom:1px solid grey ;  
}  
#table tbody tr{  
 text-align: center;  
 &:nth-child(odd){  
 background-color: rgba(129, 129, 129, 0.4);  
 &:hover{  
 background-color: rgba(161, 161, 161, 0.4);  
 }  
 }  
 &:nth-child(even){  
 &:hover{  
 background-color: #444343;  
 }  
 }  
  
}  
  
.delete-btn{  
 cursor: pointer;  
 background-image: url("../assets/svg/trash.svg");  
 width: 34px;  
 height: 34px;  
 border: none;  
}

# Ілюстрації виконання програми

## *Кнопка відкриття вікна табиці:*

## 

## *Вікно таблиці з введеними фігурами*

## 

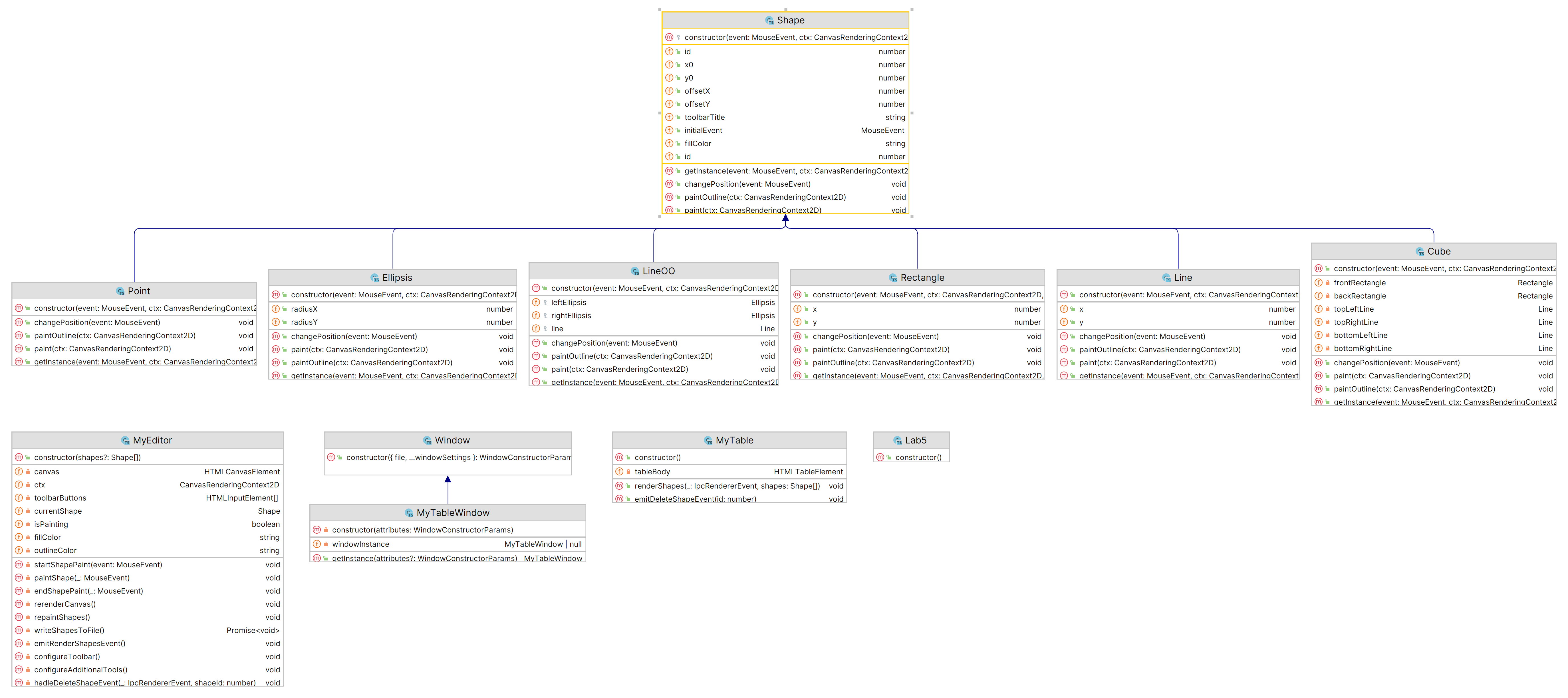
## *Видалення фігури з таблиці та полотна*

## 

## Запис фігур у файл

# 

# Діаграма класів програми

****

**Висновки**

Під час виконання лабораторної роботи було cтворено ще одне вікно редактора (вікно таблиці). Фігури у даній таблиці можна видаляти з вікна. Саме вікно було запрограмовано використовуючи патерн програмування Сінглтон. Також було додано запис фігур у файл. Лабораторна була виконана у середовищі мови Typescipt (JavaScript) на платформі Electron.js

Детальний вихідний код можна побачити на сторінці у [гітхабі](https://github.com/KPI-kujo205/2course_oop/tree/lab5)