**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 №4.**

з дисципліни

«Об’єктно орієнтоване програмування» на тему

**Вдосконалення структури коду графічного редактора об’єктів на C++**

Виконав: Перевірив:

Студент групи ІМ-22 Порєв В.М.

Куц Іван Васильович

номер у списку групи: 12

Київ 2023

# Мета

Отримати вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктноорієнтованому стилі для забезпечення зручного додавання нових типів об'єктів.

# Завдання

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab4
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. Динамічний об’єкт класу MyEditor

2. Гумовий слід — пунктирна лінія

3. Колір заливки — як у попередній роботі

4. Окрім 4 фігур з попередньої роботи, небхідно створити фігури типу “Куб” та “Гантеля”

5. Для усіх 6 фігур необхідно зробити кнопку у Тулбарі

6. Оформиит діаграму класів

# Вихідний текст програмного коду

## MyEditor.ts

import Shape from "./shapes/Shape.ts";  
import Line from "./shapes/Line.ts";  
import Point from "./shapes/Point.ts";  
import Ellipsis from "./shapes/Elipsis.ts";  
import Rectangle from "./shapes/Rectangle.ts";  
import Cube from "./shapes/Cube.ts";  
import LineOO from "./shapes/LineOO.ts";  
enum ShapeType{  
 Rectangle,  
 Ellipsis,  
 Line,  
 Point,  
 Cube,  
 LineOO  
}  
  
const TitleMapper:Record<ShapeType, string> = {  
 [ShapeType.Rectangle]:'Прямокутник',  
 [ShapeType.Ellipsis]:'Еліпс',  
 [ShapeType.Line]:'Лінія',  
 [ShapeType.Point]:'Крапка',  
 [ShapeType.Cube]:'Куб',  
 [ShapeType.LineOO]:'Гантеля'  
}  
  
class MyEditor{  
 private canvas:HTMLCanvasElement;  
 private ctx:CanvasRenderingContext2D;  
 private toolbarButtons!:HTMLInputElement[];  
 private selectedShape: ShapeType = ShapeType.Rectangle;  
 private currentShape: Shape | undefined;  
 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.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();  
  
 }  
  
 private startShapePaint(event:MouseEvent ){  
 this.isPainting=true;  
 switch (this.selectedShape){  
 case ShapeType.Rectangle:  
 this.currentShape=new Rectangle(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 case ShapeType.Ellipsis:  
 this.currentShape=new Ellipsis(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 case ShapeType.Line:  
 this.currentShape=new Line(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 case ShapeType.Point:  
 this.currentShape=new Point(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 case ShapeType.Cube:  
 this.currentShape=new Cube(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 case ShapeType.LineOO:  
 this.currentShape=new LineOO(event, this.ctx,this.fillColor,this.outlineColor);  
 break;  
 }  
 };  
 private paintShape(\_:MouseEvent) {  
 if(!this.isPainting) return;  
 this.canvas.classList.add('painting')  
 this.repaintShapes();  
 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.repaintShapes()  
  
 }  
 private repaintShapes(){  
 this.ctx.clearRect(0,0,this.ctx.canvas.width,this.ctx.canvas.height)  
 for (const shape of this.shapes) {  
 shape.paint(this.ctx);  
 }  
 }  
 private configureToolbar(){  
 this.toolbarButtons=['#ellipsis-btn','#line-btn','#rectangle-btn','#point-btn','#cube-btn','#lineOO-btn'].map((btn)=>document.querySelector(btn) as HTMLInputElement);  
 for(const button of this.toolbarButtons){  
 button.addEventListener('click',this.setActiveShapeButton.bind(this))  
 }  
 }  
 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;  
  
  
 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;  
 })  
  
 }  
 private setActiveShapeButton(event:MouseEvent){  
 const targetedButton=event.target as HTMLInputElement;  
 this.toolbarButtons.forEach(b=>b.classList.remove('selected'))  
 targetedButton.classList.add('selected')  
  
 this.selectedShape= +targetedButton.value;  
  
 const title=document.querySelector('title') as HTMLTitleElement;  
 title.innerText=TitleMapper[this.selectedShape];  
 }  
  
  
  
  
}  
export default MyEditor;

## Shape.ts

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

## Rectangle.ts

import Shape from "./Shape";  
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);  
 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])  
 }  
  
}

## Ellipsis.ts

import Shape from "./Shape";  
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);  
 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])  
  
 }  
  
}

## Line.ts

import Shape from "./Shape";  
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);  
 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();  
 }  
  
  
}

## Point.ts

import Shape from "./Shape";  
export default class Point extends Shape{  
 constructor(event:MouseEvent, ctx:CanvasRenderingContext2D,fillColor:string, outlineColor:string){  
 super(event,ctx,fillColor,outlineColor);  
 }  
 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);  
 }  
  
}

# LineOO.ts

import Shape from "./Shape.ts";  
import Ellipsis from "./Elipsis.ts";  
import Line from "./Line.ts";  
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);  
 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);  
 }  
  
}

# Cube.ts

import Shape from "./Shape.ts";  
import Rectangle from "./Rectangle.ts";  
import Line from "./Line.ts";  
  
const horizontalOffset=(line:number)=>line\*.5  
const verticalOffset=(line:number)=>line\*.33  
  
  
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);  
 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 cubeWidth=dx\*3;  
  
 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])  
  
 }  
  
  
}

# main.ts

import './style.css'  
import MyEditor from "./MyEditor.ts";  
  
const lab4 = new MyEditor();

# 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%;  
}  
  
a {  
 font-weight: 500;  
 color: #646cff;  
 text-decoration: inherit;  
}  
a:hover {  
 color: #535bf2;  
}  
  
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{  
 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");  
}  
  
  
.selected{  
 border: 2px solid springgreen !important;  
}  
  
.color-input{  
 width: 75%;  
 align-self: center;  
}

# index.html

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Прямокутник</title>  
 </head>  
 <body>  
 <header >  
 <nav class="navbar">  
 <ul>  
 <li>Тулбар <menu id="toolbar">  
 <input type="button" title="Прямокутник" id="rectangle-btn" value="0" class="selected">  
 <input type="button" title="Еліпс" id="ellipsis-btn" value="1">  
 <input type="button" title="Лінія" id="line-btn" value="2">  
 <input type="button" title="Крапка" id="point-btn" value="3">  
 <input type="button" title="Куб" id="cube-btn" value="4">  
 <input type="button" title="Гантеля" id="lineOO-btn" value="5">  
 </menu></li>  
  
 <li>Допоміжні інструменти  
 <span>  
  
 <button id="back-btn">  
 Назад  
 </button >  
 <button id="clean-btn">  
 Очистити  
 </button >  
 </span>  
  
 </li>  
  
 <li>Колір заливки  
 <input type="color" id="fill-color" value="#000000" class="color-input">  
 </li>  
  
 <li>Колір гумового сліду  
 <input type="color" id="outline-color" value="#64ff00" class="color-input">  
 </li>  
  
  
  
 </ul>  
 </nav>  
 </header>  
 <main id="app">  
 <canvas id="canvas" width="800" height="500"></canvas>  
  
  
 </main>  
 <script type="module" src="/src/main.ts"></script>  
 </body>  
</html>

# Ілюстрації виконання програми

## *Відображення прямокутника:*

## Відображення еліпсу:

## 

## *Відображення прямої:*

## Відображення крапки:

## 

## 

## Відображення куба:

## Відображення гантелі:

## 

# Діаграма класів програми

****

**Висновки**

Під час виконання лабораторної роботи було покращено вихадний код графічного редактора без зміни функціоналу, тобто був проведений рефактор коду. Було прибрано класи едіторів для певних кожно фігури. Також було введено 2 нові фігури, які базуються на інших фігурах (“Гантеля” та “Куб”).

Детальний вихідний код можна побачити на сторінці у [гітхабі](https://github.com/KPI-kujo205/2course_oop/tree/lab4)  
Переглянути робочу версію редактора онлайн [тут](https://kpi-kujo205.github.io/2course_oop/)