**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 №2**

з дисципліни

«Об’єктно орієнтоване програмування» на тему

“Розробка 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 використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів С++, запрограмувавши простий графічний редактор в об’єктно-орієнтованому стилі.

# Завдання

1. Створити у середовищі MS Visual Studio C++ проект типу Windows Desktop Application з ім’ям Lab2
2. Скомпілювати проект і отримати виконуваний файл програми
3. Перевірити роботу програми. Налагодити програму
4. Проаналізувати та прокоментувати результати та вихідний текст програми
5. Оформити звіт

**Вимоги згідно з варіантом :**

1. Динамічний масив об’єктів

2. Гумовий слід — суцільна лінія чорного кольору

3. Прямокутник по 2 протилежним кутам. Відображення прямокутника — чорний контур з кольоровим заповненням (жовтий колір)

4. Увід еліпсу від центру до одного з кутів охоплюючого прямокутника. Відображення еліпсу — чорний контур без заповнення

5. Позначка поточного типу об’єкту, що вводиться — в меню

# Вихідний текст програмного коду

## ShapesObjectEditor.tsx

import Shape from "./shapes/Shape.ts";  
import EllipsisEditor from "./editors/EllipsisEditor.ts";  
import ShapeEditor from "./editors/ShapeEditor.ts";  
import RectangleEditor from "./editors/RectangleEditor.ts";  
import LineEditor from "./editors/LineEditor.ts";  
import PointEditor from "./editors/PointEditor.ts";  
  
enum EditorType{  
 *RectangleEditor*,  
 *EllipsisEditor*,  
 *LineEditor*,  
 *PointEditor*}  
  
const EditorTypeMapper:Record<EditorType, string> = {  
 [EditorType.*RectangleEditor*]:'Прямокутник',  
 [EditorType.*EllipsisEditor*]:'Еліпс',  
 [EditorType.*LineEditor*]:'Лінія',  
 [EditorType.*PointEditor*]:'Крапка'  
}  
class ShapesObjectEditor{  
 private ctx:CanvasRenderingContext2D;  
 private editor!:ShapeEditor;  
 constructor(public shapes:Shape[]=[]) {  
 const canvas = *document*.getElementById('canvas') as HTMLCanvasElement;  
 const menu= *document*.getElementById('objects\_menu') as HTMLSelectElement;  
 this.ctx = canvas.getContext('2d') as CanvasRenderingContext2D;  
 this.configureEditor(+menu.value as EditorType)  
  
 menu.addEventListener('change',(\_)=> this.configureEditor(+menu.value as EditorType))  
 canvas.addEventListener('mousedown',(e)=>this.editor.onPaintStart(e))  
 canvas.addEventListener('mousemove', (e)=>{  
 this.repaint();  
 this.editor.onMouseMove(e)  
 })  
 canvas.addEventListener('mouseup',(e)=> {  
 this.shapes.push(this.editor.onPaintEnd(e))  
 })  
 }  
 repaint(){  
 this.ctx.clearRect(0,0,this.ctx.canvas.width,this.ctx.canvas.height)  
 for (const shape of this.shapes) {  
 shape.paint(this.ctx);  
 }  
 }  
  
 configureEditor(type:EditorType){  
 switch (type) {  
 case EditorType.*RectangleEditor*:  
 this.editor = new RectangleEditor(this.ctx);  
 break;  
 case EditorType.*EllipsisEditor*:  
 this.editor = new EllipsisEditor(this.ctx);  
 break;  
 case EditorType.*LineEditor*:  
 this.editor = new LineEditor(this.ctx);  
 break;  
 case EditorType.*PointEditor*:  
 this.editor = new PointEditor(this.ctx);  
 break;  
 }  
  
 const typeField=*document*.getElementById('object\_mode') as HTMLSpanElement;  
 typeField.innerHTML=EditorTypeMapper[type];  
 }  
  
  
}  
export default ShapesObjectEditor;

## Shape.ts

export default abstract class Shape{  
 constructor(protected x0:number,protected y0:number) {}  
 abstract paint(ctx:CanvasRenderingContext2D):void;  
}

## Rectangle.ts

import Shape from "./Shape";  
export default class Rectangle extends Shape{  
 private x:number;  
 private y:number;  
  
 constructor(x0:number,y0:number,private offsetX:number,private offsetY:number){  
 super(x0-offsetX,y0-offsetY);  
 this.x=this.x0;  
 this.y=this.y0;  
 this.offsetX=offsetX;  
 this.offsetY=offsetY;  
 }  
 changePosition(event:MouseEvent){  
 this.x=event.clientX-this.offsetX;  
 this.y=event.clientY-this.offsetY;  
 }  
  
 paint(ctx:CanvasRenderingContext2D){  
 const originalFillStyle=ctx.fillStyle;  
 ctx.fillStyle = "rgb(248,235,58)";  
 ctx.fillRect(this.x0,this.y0,this.x-this.x0,this.y-this.y0);  
 ctx.fillStyle=originalFillStyle;  
 }  
  
 paintOutline(ctx:CanvasRenderingContext2D){  
 ctx.strokeRect(this.x0,this.y0,this.x-this.x0,this.y-this.y0);  
 }  
  
}

## Ellipsis.ts

import Shape from "./Shape";  
export default class Ellipsis extends Shape{  
 private radiusX:number;  
 private radiusY:number;  
  
 constructor(x0:number,y0:number,private offsetX:number,private offsetY:number){  
 super(x0-offsetX,y0-offsetY);  
 this.radiusX=0;  
 this.radiusY=0;  
 this.offsetX=offsetX;  
 this.offsetY=offsetY;  
 }  
 changeRadius(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){  
 const originalFillStyle=ctx.fillStyle;  
 ctx.fillStyle="rgb(68,225,79)";  
 ctx.beginPath();  
 ctx.ellipse(this.x0,this.y0,this.radiusX,this.radiusY,0,0,2\**Math*.PI);  
 ctx.fill();  
 ctx.fillStyle=originalFillStyle;  
 }  
  
 paintOutline(ctx:CanvasRenderingContext2D){  
 ctx.beginPath();  
 ctx.ellipse(this.x0,this.y0,this.radiusX,this.radiusY,0,0,2\**Math*.PI);  
 ctx.stroke();  
 }  
  
}

## Line.ts

import Shape from "./Shape";  
export default class Line extends Shape{  
 private x:number;  
 private y:number;  
  
 constructor(x0:number,y0:number,private offsetX:number,private offsetY:number){  
 super(x0-offsetX,y0-offsetY);  
 this.x=this.x0;  
 this.y=this.y0;  
 this.offsetX=offsetX;  
 this.offsetY=offsetY;  
 }  
 changePosition(event:MouseEvent){  
 this.x=event.clientX-this.offsetX;  
 this.y=event.clientY-this.offsetY;  
 }  
  
 paint(ctx:CanvasRenderingContext2D){  
 ctx.beginPath();  
 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(x0:number,y0:number,private offsetX:number,private offsetY:number){  
 super(x0-offsetX,y0-offsetY);  
 this.offsetX=offsetX;  
 this.offsetY=offsetY;  
 }  
 changePosition(event:MouseEvent){  
 this.x0=event.clientX-this.offsetX;  
 this.y0=event.clientY-this.offsetY;  
 }  
  
 paint(ctx:CanvasRenderingContext2D){  
 ctx.fillRect(this.x0,this.y0,3,3);  
 }  
  
}

# ShapeEditor.ts

import Shape from "../shapes/Shape.ts";  
  
abstract class ShapeEditor{  
 abstract onPaintStart(e:MouseEvent): void;  
 abstract onMouseMove(e:MouseEvent): void;  
 abstract onPaintEnd(e:MouseEvent): Shape;  
 abstract paint(): void;  
}  
  
export default ShapeEditor

# RectangleEditor.ts

import Rectangle from "./../shapes/Rectangle.ts";  
import ShapeEditor from "./ShapeEditor.ts";  
class RectangleEditor implements ShapeEditor{  
 private startedPaint:boolean;  
 private rect:Rectangle;  
  
 constructor(private ctx:CanvasRenderingContext2D) {  
 this.ctx= ctx;  
 this.startedPaint=false;  
 this.rect=new Rectangle(0,0,0,0);  
 }  
 onPaintStart(event:MouseEvent) {  
 this.startedPaint=true;  
 this.rect=new Rectangle(event.x,event.y,this.ctx.canvas.offsetLeft,this.ctx.canvas.offsetTop);  
 }  
 onMouseMove(event:MouseEvent) {  
 if(!this.startedPaint)return;  
 this.rect.changePosition(event);  
 this.rect.paintOutline(this.ctx)  
  
 }  
 onPaintEnd(event:MouseEvent) {  
 this.startedPaint=false;  
 this.rect.changePosition(event);  
 return this.rect;  
 }  
 paint() {  
 this.rect.paint(this.ctx)  
 }  
}  
  
export default RectangleEditor;

# EllipsisEditor.ts

import Elipsis from "./../shapes/Elipsis.ts";  
import ShapeEditor from "./ShapeEditor.ts";  
class EllipsisEditor implements ShapeEditor{  
 private startedPaint:boolean;  
 private elipsis:Elipsis;  
  
 constructor(private ctx:CanvasRenderingContext2D) {  
 this.ctx= ctx;  
 this.startedPaint=false;  
 this.elipsis=new Elipsis(0,0,ctx.canvas.offsetLeft,ctx.canvas.offsetTop);  
 }  
  
 onPaintStart(event:MouseEvent) {  
 this.startedPaint=true;  
 this.elipsis=new Elipsis(event.x,event.y,this.ctx.canvas.offsetLeft,this.ctx.canvas.offsetTop);  
 }  
  
 onMouseMove(event:MouseEvent) {  
 if(!this.startedPaint)return;  
 this.elipsis.changeRadius(event);  
 this.elipsis.paintOutline(this.ctx);  
 }  
  
 onPaintEnd(event:MouseEvent) {  
 this.startedPaint=false;  
 this.elipsis.changeRadius(event);  
 return this.elipsis;  
 }  
  
 paint() {  
 this.elipsis.paint(this.ctx)  
 }  
  
  
}  
  
export default EllipsisEditor;

# LineEditor.ts

import ShapeEditor from "./ShapeEditor.ts";  
import Line from "../shapes/Line.ts";  
class LineEditor implements ShapeEditor{  
 private startedPaint:boolean;  
 private line!:Line;  
  
 constructor(private ctx:CanvasRenderingContext2D) {  
 this.ctx= ctx;  
 this.startedPaint=false;  
 }  
 onPaintStart(event:MouseEvent) {  
 this.startedPaint=true;  
 this.line=new Line(event.x,event.y,this.ctx.canvas.offsetLeft,this.ctx.canvas.offsetTop);  
 }  
 onMouseMove(event:MouseEvent) {  
 if(!this.startedPaint)return;  
 this.line.changePosition(event);  
 this.line.paint(this.ctx)  
  
 }  
 onPaintEnd(event:MouseEvent) {  
 this.startedPaint=false;  
 this.line.changePosition(event);  
 return this.line;  
 }  
 paint() {  
 this.line.paint(this.ctx)  
 }  
}  
  
export default LineEditor;

# PointEditor.ts

import ShapeEditor from "./ShapeEditor.ts";  
import Point from "../shapes/Point.ts";  
class PointEditor implements ShapeEditor{  
 private startedPaint:boolean;  
 private point!:Point;  
  
 constructor(private ctx:CanvasRenderingContext2D) {  
 this.ctx= ctx;  
 this.startedPaint=false;  
 }  
 onPaintStart(event:MouseEvent) {  
 this.startedPaint=true;  
 this.point=new Point(event.x,event.y,this.ctx.canvas.offsetLeft,this.ctx.canvas.offsetTop);  
 }  
 onMouseMove(event:MouseEvent) {  
 if(!this.startedPaint)return;  
 this.point.changePosition(event);  
 this.point.paint(this.ctx)  
  
 }  
 onPaintEnd(event:MouseEvent) {  
 this.startedPaint=false;  
 this.point.changePosition(event);  
 return this.point;  
 }  
 paint() {  
 this.point.paint(this.ctx)  
 }  
}  
  
export default PointEditor;

# main.ts

import './style.css'  
import ShapesObjectsEditor from './ShapesObjectsEditor.ts'  
  
const lab2 = new ShapesObjectsEditor();

# index.html

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <link rel="icon" type="image/svg+xml" href="/vite.svg" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Vite + TS</title>  
 </head>  
 <body>  
 <header >  
 <nav class="navbar">  
 <ul>  
 <li>  
 <label for="file">Файл</label>  
 <select id="file">  
 <option value="0">Створити файл</option>  
 <option value="1">Зберегти файл</option>  
 <option value="2">Відкрити файл</option>  
 </select>  
 </li>  
 <li>  
 <label for="objects\_menu">Об'єкти</label>  
 <select id="objects\_menu">  
 <option value="0">Прямокутник</option>  
 <option value="1">Еліпс</option>  
 <option value="2">Лінія</option>  
 <option value="3"> Крапка</option>  
 </select>  
 </li>  
 <li>  
 <label for="objects\_menu">Довідка</label>  
 <select id="faq">  
 <option value="0">Прямокутник</option>  
 <option value="1"> Еліпс</option>  
 <option value="2">Лінія</option>  
 <option value="3"> Крапка</option>  
 </select>  
 </li>  
 <li>Режим вводу об'єкту: <span id="object\_mode"></span></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 еліпса:*

## 

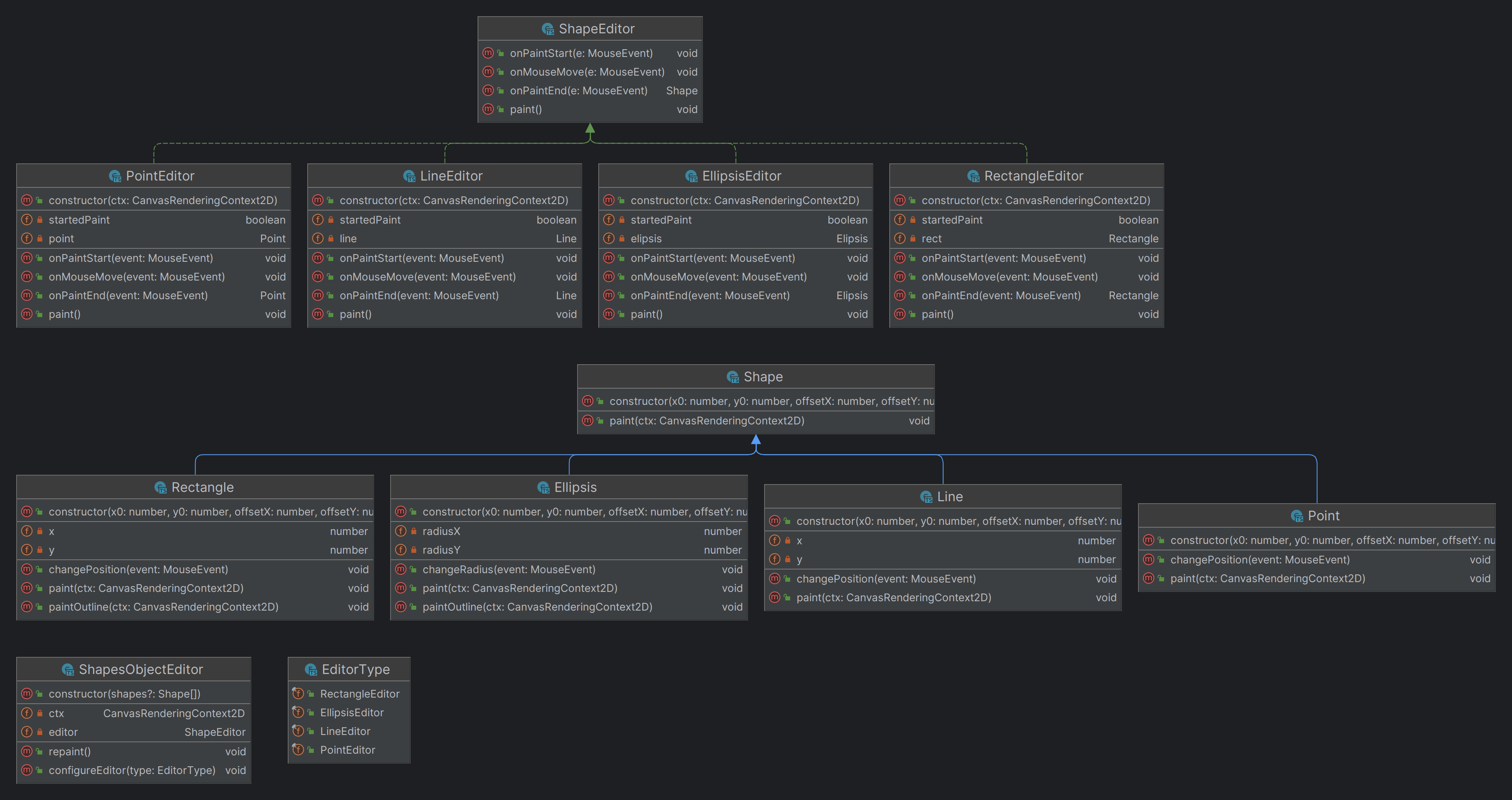
## Відображення лінії:

## Відображення крапки:

## 

## 

# Діаграма класів програми

****

**Висновки**

Під час виконання лабораторної роботи я запрограмував простий графічний редактор у стилі ООП на мові TypeScript, для цього я використовував основні поняття даного стилю програмування: поліморфізм, успадкування та асбтракцію.

Детальний вихідний код можна побачити на сторінці у [гітхабі](https://github.com/KPI-kujo205/2course_oop/tree/main/lab2):