**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

Розробка інтерфейсу користувача 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 Win32 з ім’ям Lab3.
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 масив об’єктів

2. Гумовий слід — суцільна лінія червоного кольору

3. Прямокутник від центру до одного з кутів для, чорний контур без заповнення

4. Увід еліпсу по двом протилежним кутам охоплюючого прямокутника. Відображення еліпсу — чорний контур з жовтим заповненням

5. Позначка поточного типу об’єкту, що вводиться — у назві вікна

# 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* HTMLInputElement:Record<EditorType, string> = {  
 [EditorType.RectangleEditor]:'Прямокутник',  
 [EditorType.EllipsisEditor]:'Еліпс',  
 [EditorType.LineEditor]:'Лінія',  
 [EditorType.PointEditor]:'Крапка'  
}  
  
  
*class* ShapesObjectEditor{  
 *private* ctx:CanvasRenderingContext2D;  
 *private* editor!:ShapeEditor;  
 *private* toolbarButtons!:HTMLInputElement[];  
 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))  
 })  
 *this*.configureToolbar()  
 *this*.configureAdditionalTools();  
  
 }  
 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* title=document.querySelector('title') as HTMLTitleElement;  
 title.innerText=HTMLInputElement[type];  
 }  
  
 configureToolbar(){  
 *this*.toolbarButtons=['#ellipsis-btn','#line-btn','#rectangle-btn','#point-btn'].map((btn)=>document.querySelector(btn) as HTMLInputElement);  
  
 *//for styling  
 for*(*const* button of *this*.toolbarButtons){  
 button.addEventListener('click',*this*.onToolbarButtonClick.bind(*this*))  
 }  
  
  
 }  
 configureAdditionalTools(){  
 *const* cleanButton=document.querySelector('#clean-btn') as HTMLButtonElement;  
 *const* backButton=document.querySelector('#back-btn') as HTMLButtonElement;  
  
 cleanButton.addEventListener('click',()=>{  
 *this*.shapes=[];  
 *this*.repaint()  
 })  
  
 backButton.addEventListener('click',()=>{  
 *this*.shapes.pop();  
 *this*.repaint()  
 })  
  
 }  
  
 onToolbarButtonClick(event:MouseEvent){  
 *const* targetedButton=event.target as HTMLInputElement;  
 *this*.toolbarButtons.forEach(b=>b.classList.remove('selected'))  
 targetedButton.classList.add('selected')  
 *this*.configureEditor(+targetedButton.value)  
 }  
  
  
  
  
}  
*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*,offsetX:*number*,offsetY:*number*){  
 *super*(x0-offsetX,y0-offsetY,offsetX,offsetY);  
 *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){  
 *const* deltaX=*this*.x-*this*.x0;  
 *const* deltaY=*this*.y-*this*.y0;  
  
 ctx.strokeRect(*this*.x0-deltaX,*this*.y0-deltaY,2\*deltaX,2\*deltaY);  
 }  
  
 paintOutline(ctx:CanvasRenderingContext2D){  
 *const* deltaX=*this*.x-*this*.x0;  
 *const* deltaY=*this*.y-*this*.y0;  
  
  
 *const* originalStrokeStyle=ctx.fillStyle;  
 ctx.strokeStyle = "rgb(248,32,87)";  
 ctx.strokeRect(*this*.x0-deltaX,*this*.y0-deltaY,2\*deltaX,2\*deltaY);  
  
 ctx.strokeStyle=originalStrokeStyle;  
 }  
  
}

## Ellipsis.ts

*import* Shape *from* "./Shape";  
*export default class* Ellipsis *extends* Shape{  
 *private* x:*number*;  
 *private* y:*number*;  
  
 *constructor*(x0:*number*,y0:*number*,offsetX:*number*, offsetY:*number*){  
 *super*(x0-offsetX,y0-offsetY,offsetX,offsetX);  
 *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){  
 *const* originalFillStyle=ctx.fillStyle;  
 *const* originalStrokeStyle=ctx.strokeStyle;  
  
 ctx.fillStyle="rgb(238,225,45)";  
 ctx.strokeStyle="rgb(0,0,0)"  
  
 *const* centerX=(*this*.x+*this*.x0)/2  
 *const* centerY=(*this*.y+*this*.y0)/2;  
 *const* radiusX=Math.abs((*this*.x-*this*.x0)/2)  
 *const* radiusY=Math.abs((*this*.y-*this*.y0)/2)  
  
 ctx.beginPath();  
 ctx.ellipse(centerX,centerY,radiusX,radiusY,0,0,2\*Math.PI);  
 ctx.fill();  
 ctx.stroke();  
  
  
 ctx.fillStyle=originalFillStyle  
 ctx.strokeStyle=originalStrokeStyle  
 }  
  
 paintOutline(ctx:CanvasRenderingContext2D){  
 *const* originalStrokeStyle=ctx.fillStyle;  
  
 *const* centerX=(*this*.x+*this*.x0)/2  
 *const* centerY=(*this*.y+*this*.y0)/2;  
 *const* radiusX=Math.abs((*this*.x-*this*.x0)/2)  
 *const* radiusY=Math.abs((*this*.y-*this*.y0)/2)  
  
 ctx.beginPath();  
 ctx.ellipse(centerX,centerY,radiusX,radiusY,0,0,2\*Math.PI);  
  
 ctx.strokeStyle = "rgb(248,32,87)";  
 ctx.stroke();  
 ctx.strokeStyle=originalStrokeStyle;  
 }  
  
}

## 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.changePosition(event);  
 *this*.elipsis.paintOutline(*this*.ctx);  
 }  
  
 onPaintEnd(event:MouseEvent) {  
 *this*.startedPaint=*false*;  
 *this*.elipsis.changePosition(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* lab3 = *new* ShapesObjectsEditor();

# index.html

<!doctype *html*>  
<html *lang*="en">  
 <head>  
 <meta *charset*="UTF-8" />  
 <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0" />  
 <title>Лабораторна робота 2</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>Тулбар <menu *id*="toolbar">  
 <input *type*="button" *title*="Прямокутник" *id*="rectangle-btn" *value*="0">  
 <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">  
 </menu></li>  
  
 <li>Допоміжні інструменти  
 <span>  
  
 <button *id*="back-btn">  
 Назад  
 </button >  
 <button *id*="clean-btn">  
 Очистити  
 </button >  
 </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 було покращено графічний редактор, розроблений у попередній лабоарторній роботі, було додано тулбар з кнопками для вибору фігур, також було додано кнопки “Назад” тa “Очистити”. У цій роботі, так як і у попередній, було викоростано поняття абстракції, інкапсуляцій та поліморфізму

Детальний вихідний код можна побачити на сторінці у [гітхабі](https://github.com/KPI-kujo205/2course_oop/tree/lab3/lab2)