|  |  |
| --- | --- |
| **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» | |
| Виконав: cтудент групи ІМ-34 **Нетяга Максим Костянтинович** номер у списку групи: 14 | Перевірив: **Порєв В. М.** |
| Київ 2024 | |

# Вихідний код

## application.py

import tkinter as tk

from .editor\_canvas import EditorCanvas, ShapeNames

class Application(tk.Tk):

    def \_\_init\_\_(self, screenName=None, baseName=None,

                 className="oop\_lab2", useTk=True, sync=False,

                 use=None):

        super().\_\_init\_\_(screenName, baseName, className, useTk, sync, use)

        self.resizable(False, False)

        self.geometry("500x500")

        self.\_editor\_canvas = EditorCanvas(self)

        self.\_menubar = \_MenuBar(self)

        # Application config

        self.config(menu=self.\_menubar, border=5)

        # Placing widgets

        self.\_editor\_canvas.pack(fill=tk.BOTH, expand=True)

        # self.menubar.grid(row=0, column=0)

    def select\_shape(self, shape: ShapeNames):

        self.\_editor\_canvas.select\_shape(shape)

    def clear\_canvas(self):

        self.\_editor\_canvas.clear\_canvas()

class \_MenuBar(tk.Menu):

    def \_\_init\_\_(self, parent: Application):

        super().\_\_init\_\_(parent)

        file\_menu = tk.Menu(self, tearoff=0)

        file\_menu.add\_command(

            label="Очистити",

            command=parent.clear\_canvas

        )

        object\_menu = tk.Menu(self, tearoff=0)

        object\_menu.add\_radiobutton(

            label="Крапка",

            command=(lambda: parent.select\_shape("Dot"))

        )

        object\_menu.add\_radiobutton(

            label="Лінія",

            command=(lambda: parent.select\_shape("Line"))

        )

        object\_menu.add\_radiobutton(

            label="Прямокутник",

            command=(lambda: parent.select\_shape("Rectangle"))

        )

        object\_menu.add\_radiobutton(

            label="Еліпс",

            command=(lambda: parent.select\_shape("Ellipse"))

        )

        info\_menu = tk.Menu(self, tearoff=0)

        self.add\_cascade(label="Файл", menu=file\_menu)

        self.add\_cascade(label="Об’єкти", menu=object\_menu)

        self.add\_cascade(label="Довідка", menu=info\_menu)

## editor\_canvas.py

from typing import Literal

from abc import ABC, abstractmethod

import tkinter as tk

ShapeNames = Literal["Dot", "Line", "Rectangle", "Ellipse"]

class EditorCanvas(tk.Canvas):

    @staticmethod

    def center\_coords\_to\_boundary(x0, y0, x1, y1):

        return (x0 - (x1 - x0)/2, y0 - (y1 - y0)/2, x1, y1)

    def \_\_init\_\_(self, parent):

        super().\_\_init\_\_(parent, bg="white")

        self.\_shape\_to\_draw: ShapeNames = None

        self.\_editable\_shape: \_Shape = None

        self.shapes = {}

        # Bind mouse events

        self.bind("<ButtonPress-1>", self.on\_mouse\_down)

        self.bind("<B1-Motion>", self.on\_mouse\_drag)

        self.bind("<ButtonRelease-1>", self.on\_mouse\_up)

    def select\_shape(self, shape\_name: ShapeNames):

        self.\_shape\_to\_draw = shape\_name

    def on\_mouse\_down(self, event):

        if not self.\_shape\_to\_draw:

            return

        self.\_editable\_shape = \_SHAPES[self.\_shape\_to\_draw]()

        self.\_editable\_shape.start\_drawing(self, event)

    def on\_mouse\_drag(self, event):

        if not self.\_editable\_shape:

            return

        self.\_editable\_shape.draw(self, event)

    def on\_mouse\_up(self, event):

        if not self.\_editable\_shape:

            return

        self.\_editable\_shape.stop\_drawing(self, event)

        self.\_editable\_shape = None

    def clear\_canvas(self):

        self.delete("all")

        self.shapes.clear()

class \_Shape(ABC):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.shape\_id = None

        self.placeholder\_id = None

        self.start\_x = None

        self.start\_y = None

    @abstractmethod

    def start\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        self.start\_x, self.start\_y = event.x, event.y

        self.placeholder\_id = canvas.create\_rectangle(

            -1, -1, -1, -1,

            outline="blue"

        )

    @abstractmethod

    def draw(self, canvas: EditorCanvas, event: tk.Event):

        pass

    @abstractmethod

    def stop\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        canvas.delete(self.placeholder\_id)

        self.placeholder\_id = None

        canvas.shapes[self.shape\_id] = self

class \_Dot(\_Shape):

    def start\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().start\_drawing(canvas, event)

        self.shape\_id = canvas.create\_oval(

            event.x - 1, event.y - 1,

            event.x + 1, event.y + 1,

            fill="black"

        )

    def draw(self, canvas: EditorCanvas, event):

        super().draw(canvas, event)

    def stop\_drawing(self, canvas: EditorCanvas, event):

        super().stop\_drawing(canvas, event)

class \_Line(\_Shape):

    def start\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().start\_drawing(canvas, event)

        self.shape\_id = canvas.create\_line(-1, -1, -1, -1)

    def draw(self, canvas: EditorCanvas, event: tk.Event):

        super().draw(canvas, event)

        canvas.coords(

            self.placeholder\_id, self.start\_x, self.start\_y,

            event.x, event.y

        )

    def stop\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().stop\_drawing(canvas, event)

        canvas.coords(

            self.shape\_id, self.start\_x, self.start\_y,

            event.x, event.y

        )

class \_Rectangle(\_Shape):

    def start\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().start\_drawing(canvas, event)

        self.start\_x, self.start\_y = event.x, event.y

        self.shape\_id = canvas.create\_rectangle(-1, -1, -1, -1)

    def draw(self, canvas: EditorCanvas, event: tk.Event):

        super().draw(canvas, event)

        canvas.coords(

            self.placeholder\_id, self.start\_x, self.start\_y,

            event.x, event.y

        )

    def stop\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().stop\_drawing(canvas, event)

        canvas.coords(

            self.shape\_id, self.start\_x, self.start\_y,

            event.x, event.y

        )

class \_Ellipse(\_Shape):

    def start\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().start\_drawing(canvas, event)

        self.shape\_id = canvas.create\_oval(

            -1, -1, -1, -1,

            fill="lightgreen"

        )

    def draw(self, canvas: EditorCanvas, event: tk.Event):

        super().draw(canvas, event)

        canvas.coords(

            self.placeholder\_id,

            \*EditorCanvas.center\_coords\_to\_boundary(

                self.start\_x, self.start\_y, event.x, event.y

            )

        )

    def stop\_drawing(self, canvas: EditorCanvas, event: tk.Event):

        super().stop\_drawing(canvas, event)

        canvas.coords(

            self.shape\_id,

            \*EditorCanvas.center\_coords\_to\_boundary(

                self.start\_x, self.start\_y, event.x, event.y

            )

        )

\_SHAPES = {

    "Dot": \_Dot,

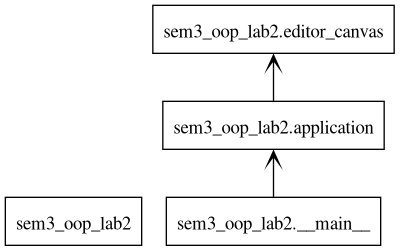
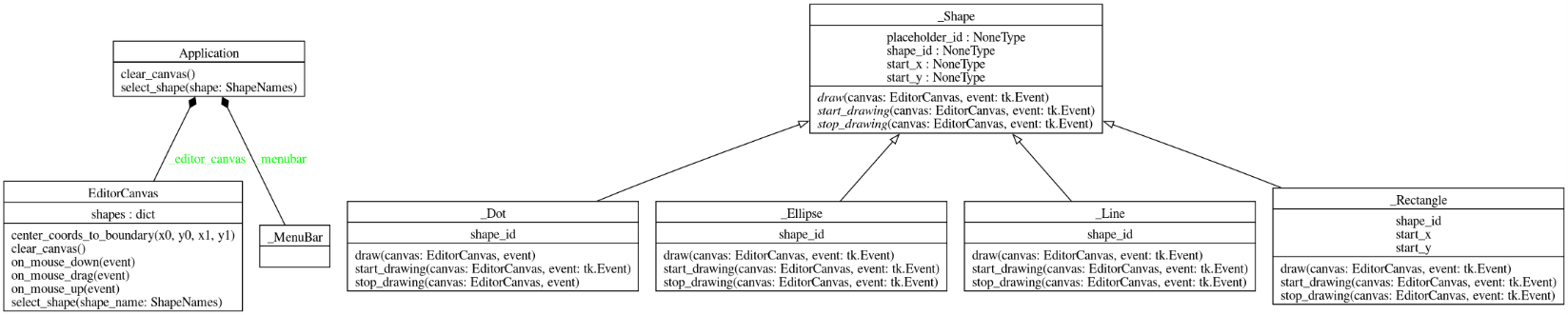
    "Line": \_Line,

    "Rectangle": \_Rectangle,

    "Ellipse": \_Ellipse,

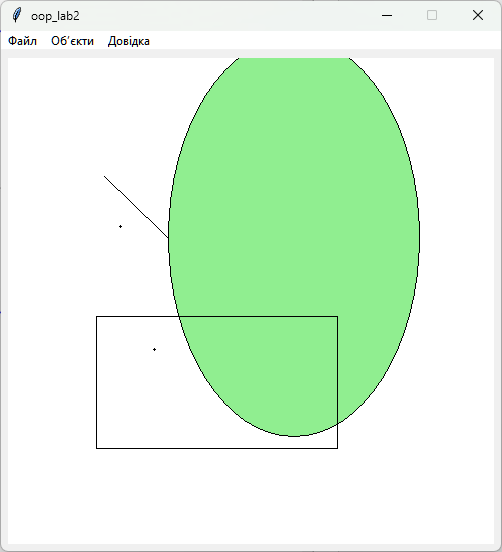
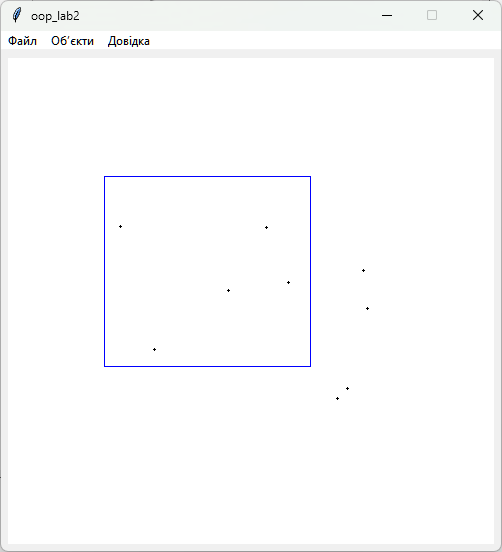
}

# Діаграми залежностей



# Скріншоти

# 



# Висновки

В ході виконання лабораторної роботи було розроблено програму для реалізації механізму малювання графічних фігур (точок, ліній, прямокутників, еліпсів) з використанням патерну "Стратегія". Було детально вивчено та застосовано концепції об’єктно-орієнтованого програмування, такі як інкапсуляція, поліморфізм та успадкування. Використання патерну "Стратегія" дозволило гнучко змінювати алгоритми малювання фігур, не змінюючи основний код програми. Також було продемонстровано можливість легко додавати нові стратегії малювання, що забезпечує масштабованість і гнучкість розширення функціоналу програми.

Таким чином, виконана робота дала змогу закріпити теоретичні знання про патерни проектування та застосувати їх на практиці для вирішення задач графічного інтерфейсу в Python за допомогою бібліотеки Tkinter.