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»

Виконала:

студентка групи ІМ-43
Хубеджева Єлизавета Павлівна
номер у списку групи: 26

Перевірив:

Порєв В. М.

Київ 2025

Завдання

- Так як варіант 26, то об’єкт класу MyEditor запрограмований на основі класичної реалізації Singleton.
- Статичний масив
- **N** = 126 - кількість елементів масиву
- **Гумовий слід:** пунктирна лінія чорного кольору
- **Прямокутник:**

  Малювання від центру до одного з кутів

  Чорний контур прямокутника без заповнення
- **Еліпс:**

  Малювання по двом протилежним кутам охоплюючого прямокутника

  Чорний контур з кольоровим заповненням

  Помаранчевий колір заповнення
- **Лінія з кружечками та каркас куба**
- **Позначка поточного типу об’єкту, що вводиться** в заголовку вікна
- **Обєкт класу MyEditor:** динамічний
- Такожбули виконані завдання на заохочувальні бали:

1) Якщо у вікні таблиці буде передбачено, щоб користувач міг виділити курсором рядок таблиці і відповідний об’єкт буде якось виділятися на зображенні у головному вікні.

2) Якщо у вікні таблиці користувач може виділити курсором рядок таблиці і відповідний об’єкт буде вилучено з масиву об’єктів.

Текст головного файлу програми

main.py:

```
import tkinter as tk
from app import DrawingApp

def main():
    root = tk.Tk()
    app = DrawingApp(root)
    app.run()

if __name__ == "__main__":
    main()
```

Тексти модульних файлів програми

app.py:

```
import tkinter as tk
from tkinter import messagebox
from toolbar import Toolbar
from my_editor import MyEditor
from my_table import MyTable

class DrawingApp:

    def __init__(self, root):
        self.root = root
        self.base_title = "Лабораторна робота 5. Хубеджева Єлизавета"
        self.root.title(self.base_title)
        self.root.geometry("950x600")

        self.current_tool = 'point'

        self.table_window = None

        self.create_menu()
        self.create_toolbar()
        self.create_canvas()

        self.my_editor = MyEditor(
            self.canvas,
            self.update_status,
            self.on_shape_update
        )

        self.create_status_bar()

        self.select_tool('point')

        self.canvas.bind("<Button-1>", self.my_editor.on_mouse_down)
        self.canvas.bind("<B1-Motion>", self.my_editor.on_mouse_move)
        self.canvas.bind("<ButtonRelease-1>", self.my_editor.on_mouse_up)

        self.root.protocol("WM_DELETE_WINDOW", self.on_program_exit)
```

```
    def create_menu(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)

        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Файл", menu=file_menu)
        file_menu.add_command(label="Очистити",
command=self.clear_canvas)
        file_menu.add_separator()
        file_menu.add_command(label="Вихід",
command=self.on_program_exit)

        self.objects_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Об'єкти", menu=self.objects_menu)

        menu_items = {
            "Точка": "point",
            "Лінія": "line",
            "Прямокутник": "rect",
            "Еліпс": "ellipse",
            "Лінія з кружечками": "line_oo",
            "Куб": "cube"
        }

        for label, tool in menu_items.items():
            self.objects_menu.add_command(
                label=label,
                command=lambda t=tool: self.select_tool(t)
            )

        table_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Таблиця", menu=table_menu)
        table_menu.add_command(label="Показати / Сховати",
command=self.toggle_table_window)

        help_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Довідка", menu=help_menu)
        help_menu.add_command(label="Про програму",
command=self.show_about)
```

```
    def create_toolbar(self):
        self.toolbar = Toolbar(self.root, self.select_tool)

    def create_canvas(self):
        self.canvas = tk.Canvas(
            self.root,
            bg="white",
            cursor="crosshair"
        )
        self.canvas.pack(fill=tk.BOTH, expand=True)

    def create_status_bar(self):
        shape_count = self.my_editor.get_shape_count()
        self.status_bar = tk.Label(
            self.root,
            text=f"Об'єктів: {shape_count}/{MyEditor.MAX_SHAPES}",
            bd=1,
            relief=tk.SUNKEN,
            anchor=tk.W
        )
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)

    def select_tool(self, tool):
        self.my_editor.unhighlight_all()

        self.current_tool = tool
        self.my_editor.select_tool(tool)
        self.toolbar.set_active_tool(tool)
        self.update_status()

    def clear_canvas(self):
        if self.my_editor.get_shape_count() > 0:
             if messagebox.askyesno("Підтвердження", "Ви впевнені, що
хочете очистити полотно?"):
                self.my_editor.clear()

    def update_status(self):
        shape_count = self.my_editor.get_shape_count()
        self.status_bar.config(
            text=f"Об'єктів: {shape_count}/{MyEditor.MAX_SHAPES}"
```

```
        )

        tool_names = {
            'point': 'Точка',
            'line': 'Лінія',
            'rect': 'Прямокутник',
            'ellipse': 'Еліпс',
            'line_oo': 'Лінія з кружечками',
            'cube': 'Куб'
        }
        current_tool_name = tool_names.get(self.current_tool, "Невідомо")

        self.root.title(f"{self.base_title} - [{current_tool_name}]")

    def show_about(self):
        messagebox.showinfo(
            "Про програму",
            "Програма для малювання геометричних фігур\n\n"
            "Автор: Хубеджева Єлизавета\n\n"
            "Підтримувані фігури:\n"
            "• Точка\n"
            "• Лінія\n"
            "• Прямокутник\n"
            "• Еліпс\n"
            "• Лінія з кружечками\n"
            "• Каркас кубу\n\n"
            "Максимальна кількість об'єктів: 126"
        )



    def _get_table_window(self):
        if self.table_window is None or not
self.table_window.window.winfo_exists():
            self.table_window = MyTable(
                self.root,
                on_row_select_callback=self.on_table_row_select,
                on_row_double_click_callback=self.on_table_row_double_cl
ick
            )

            self.on_shape_update(None)
```

```
        return self.table_window

    def toggle_table_window(self):
        table = self._get_table_window()
        if table.window.winfo_viewable():
            table.hide()
            self.my_editor.unhighlight_all()
        else:
            table.show()

    def on_shape_update(self, shape):
        if self.table_window is None or not
self.table_window.window.winfo_exists():
            return

        if shape:
            self.table_window.add_row(
                shape.__class__.__name__,
                shape.x1, shape.y1, shape.x2, shape.y2
            )
        else:
            self.table_window.clear_table()
            if self.my_editor.shapes:
                for i in range(self.my_editor.shape_count):
                    s = self.my_editor.shapes[i]
                    self.table_window.add_row(
                        s.__class__.__name__,
                        s.x1, s.y1, s.x2, s.y2
                    )

    def on_table_row_select(self, index):
        self.my_editor.highlight_shape_at(index)

    def on_table_row_double_click(self, index):
        if messagebox.askyesno("Підтвердження", "Ви впевнені, що хочете
видалити цей об'єкт?"):
            self.my_editor.delete_shape_at(index)

    def on_program_exit(self):
        if self.table_window and self.table_window.window.winfo_exists():
            self.table_window.destroy_window()
```

```
        self.root.quit()

    def run(self):
        self.root.mainloop()
```

toolbar.py:

```
import tkinter as tk

class Tooltip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip_window = None
        self.widget.bind("<Enter>", self._show_tip)
        self.widget.bind("<Leave>", self._hide_tip)

    def _show_tip(self, event=None):
        x, y, _, _ = self.widget.bbox("insert")
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 25
        self.tooltip_window = tk.Toplevel(self.widget)
        self.tooltip_window.wm_overrideredirect(True)
        self.tooltip_window.wm_geometry(f"+{x}+{y}")
        label = tk.Label(
            self.tooltip_window, text=self.text, justify='left',
            background="#ffffe0", relief='solid', borderwidth=1,
        )
        label.pack(ipadx=1)

    def _hide_tip(self, event=None):
        if self.tooltip_window:
            self.tooltip_window.destroy()
        self.tooltip_window = None

class Toolbar:
    def __init__(self, root, select_tool_callback):
        self.frame = tk.Frame(root, bd=1, relief=tk.RAISED)
        self.select_tool_callback = select_tool_callback
        self.button_images = {}
        self.buttons = {}

        buttons_data = {
            'point': ("./icons/point.png", "Створити точку"),
            'line': ("./icons/line.png", "Намалювати лінію"),
```

```
            'rect': ("./icons/rect.png", "Намалювати прямокутник"),
            'ellipse': ("./icons/ellipse.png", "Намалювати еліпс"),
            'line_oo': ("./icons/line_oo.png", "Намалювати лінію з
кружечками"),
            'cube': ("./icons/cube.png", "Намалювати каркас кубу"),
        }

        img_scale = 22

        for tool, (img_file, tooltip_text) in buttons_data.items():
            try:
                original_img = tk.PhotoImage(file=img_file)
                resized_img = original_img.subsample(img_scale,
img_scale)
                self.button_images[tool] = resized_img

                button = tk.Button(
                    self.frame,
                    image=resized_img,
                    command=lambda t=tool: self.select_tool_callback(t),
                    relief=tk.RAISED
                )
                button.pack(side=tk.LEFT, padx=2, pady=2)
                Tooltip(button, tooltip_text)
                self.buttons[tool] = button
            except tk.TclError:
                print(f"Помилка: Файл зображення '{img_file}' не
знайдено.")

        self.frame.pack(side=tk.TOP, fill=tk.X)

    def set_active_tool(self, active_tool):
        for tool, button in self.buttons.items():
            if tool == active_tool:
                button.config(relief=tk.SUNKEN)
            else:
                button.config(relief=tk.RAISED)
```

my-editor.py:

```
import tkinter as tk
from tkinter import messagebox
from shapes import Shape, PointShape, LineShape, RectShape, EllipseShape, LineOOShape, CubeShape


class MyEditor:

    MAX_SHAPES = 126
    LOG_FILE = "shapes_log.txt"

    _instance = None
    _initialized = False

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(MyEditor, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self, canvas=None, status_update_callback=None, shape_add_callback=None):
        if self._initialized:
            return

        self.canvas = canvas
        self.status_update_callback = status_update_callback
        self.shape_add_callback = shape_add_callback

        self.shapes = None
        self.shape_count = 0

        self.current_tool = 'point'
        self.is_drawing = False
        self.start_x = 0
        self.start_y = 0

        self.temp_shape = None
        self.highlighted_index = None
```

```
        self._initialized = True

        self._clear_log_file()

    def _ensure_array_exists(self):
        if self.shapes is None:
            self.shapes = [None] * self.MAX_SHAPES

    def select_tool(self, tool):
        self.current_tool = tool
        self.unhighlight_all()

    def on_mouse_down(self, event):
        self._ensure_array_exists()

        if self.shape_count >= self.MAX_SHAPES:
            messagebox.showwarning("Увага", "Досягнуто максимальну
кількість об'єктів (126)")
            return

        self.unhighlight_all()
        self.is_drawing = True
        self.start_x = event.x
        self.start_y = event.y

    def on_mouse_move(self, event):
        if not self.is_drawing:
            return

        self.canvas.delete("preview")
        self.temp_shape = self._create_shape_instance(event)

        if self.temp_shape:
            self.temp_shape.show(self.canvas, is_preview=True,
is_highlighted=False)

    def on_mouse_up(self, event):
        if not self.is_drawing:
            return

        self.is_drawing = False
```

```
        self.canvas.delete("preview")

        final_shape = self._create_shape_instance(event)

        if final_shape:
            self.add_shape(final_shape)

        self.temp_shape = None

    def _create_shape_instance(self, event):
        x1, y1 = self.start_x, self.start_y
        x2, y2 = event.x, event.y

        if self.current_tool == 'point':
            return PointShape(x1, y1, x1, y1)
        elif self.current_tool == 'line':
            return LineShape(x1, y1, x2, y2)
        elif self.current_tool == 'rect':
            return RectShape(x1, y1, x2, y2)
        elif self.current_tool == 'ellipse':
            return EllipseShape(x1, y1, x2, y2)
        elif self.current_tool == 'line_oo':
            return LineOOShape(x1, y1, x2, y2)
        elif self.current_tool == 'cube':
            return CubeShape(x1, y1, x2, y2)
        return None

    def add_shape(self, shape):
        self._ensure_array_exists()

        if self.shape_count < self.MAX_SHAPES:
            self.highlighted_index = None
            self.shapes[self.shape_count] = shape
            self.shape_count += 1
            shape.show(self.canvas, is_preview=False,
is_highlighted=False)

            self.status_update_callback()
            self._log_shape_to_file(shape)
            self.shape_add_callback(shape)
```

```
    def _log_shape_to_file(self, shape):

        shape_name = shape.__class__.__name__

        try:
            with open(self.LOG_FILE, "a", encoding="utf-8") as f:
                f.write(f"{shape_name}\t{shape.x1}\t{shape.y1}\t{shape.x
2}\t{shape.y2}\n")
        except IOError as e:
            print(f"Помилка запису у файл '{self.LOG_FILE}': {e}")

    def _clear_log_file(self):
        try:
            with open(self.LOG_FILE, "w", encoding="utf-8") as f:
                f.write("")
        except IOError as e:
            print(f"Помилка очищення файлу '{self.LOG_FILE}': {e}")

    def get_shape_count(self):
        if self.shapes is None:
            return 0
        return self.shape_count

    def clear(self):
        self.canvas.delete("shape")
        self.highlighted_index = None
        self.shapes = None
        self.shape_count = 0



        self.status_update_callback()
        self._clear_log_file()
        self.shape_add_callback(None)

    def redraw_all_shapes(self):
        self.canvas.delete("shape")
        if self.shapes is None:
            return
```

```
        for i in range(self.shape_count):
            is_highlighted = (i == self.highlighted_index)
            self.shapes[i].show(self.canvas, is_preview=False,
is_highlighted=is_highlighted)

    def highlight_shape_at(self, index):
        if not (0 <= index < self.shape_count):
            return

        if self.highlighted_index == index:
            return

        self.highlighted_index = index
        self.redraw_all_shapes()

    def unhighlight_all(self):
        if self.highlighted_index is None:
            return

        self.highlighted_index = None
        self.redraw_all_shapes()

    def delete_shape_at(self, index):
        if not (0 <= index < self.shape_count):
            return

        for i in range(index, self.shape_count - 1):
            self.shapes[i] = self.shapes[i+1]

        self.shapes[self.shape_count - 1] = None
        self.shape_count -= 1
        self.highlighted_index = None
        self.redraw_all_shapes()
        self.status_update_callback()
        self._regenerate_log_file()
        self.shape_add_callback(None)

    def _regenerate_log_file(self):
        self._clear_log_file()
        if self.shapes is None:
```

```
        return
    for i in range(self.shape_count):
        self._log_shape_to_file(self.shapes[i])
```

shapes.py:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, x1, y1, x2, y2, color="black", width=2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.color = color
        self.width = width

    @abstractmethod
    def show(self, canvas, is_preview=False, is_highlighted=False):
        pass

    def _get_style(self, is_preview, is_highlighted, default_style):
        if is_preview:
            return {"outline": "black", "fill": None, "dash": (4, 2)}

        if is_highlighted:
            highlight_style = default_style.copy()
            highlight_style["outline"] = "red"
            highlight_style["width"] = 3
            if "fill" in highlight_style and highlight_style["fill"]:
                highlight_style["fill"] = ""
            return highlight_style

        return default_style

    def _get_tags(self, is_preview):
        return "preview" if is_preview else "shape"

class PointShape(Shape):
    def show(self, canvas, is_preview=False, is_highlighted=False):
        radius = 3

        style = self._get_style(is_preview, is_highlighted, {
            "fill": self.color,
```

```
            "outline": self.color
        })

        if is_highlighted:
            radius = 4
            style["fill"] = "red"

        canvas.create_oval(
            self.x1 - radius, self.y1 - radius,
            self.x1 + radius, self.y1 + radius,
            **style,
            tags=self._get_tags(is_preview)
        )

class LineShape(Shape):
    def show(self, canvas, is_preview=False, is_highlighted=False):
        style = self._get_style(is_preview, is_highlighted, {
            "fill": self.color,
            "width": self.width
        })

        if "outline" in style:
            style["fill"] = style.pop("outline")

        if is_preview:
            style["width"] = 1

        if is_highlighted:
            style["fill"] = "red"

        canvas.create_line(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class RectShape(Shape):
    def show(self, canvas, is_preview=False, is_highlighted=False):
        style = self._get_style(is_preview, is_highlighted, {
            "fill": None,
```

```
            "outline": self.color,
            "width": self.width
        })

        canvas.create_rectangle(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class EllipseShape(Shape):
    def show(self, canvas, is_preview=False, is_highlighted=False):
        style = self._get_style(is_preview, is_highlighted, {
            "fill": "orange",
            "outline": self.color,
            "width": self.width
        })

        if is_highlighted:
            style["fill"] = None

        canvas.create_oval(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class LineOOShape(LineShape, EllipseShape):

    CIRCLE_RADIUS = 7

    def __init__(self, x1, y1, x2, y2):
        Shape.__init__(self, x1, y1, x2, y2)
        self.radius = self.CIRCLE_RADIUS

    def show(self, canvas, is_preview=False, is_highlighted=False):
        LineShape.show(self, canvas, is_preview, is_highlighted)
        orig_x1, orig_y1, orig_x2, orig_y2 = self.x1, self.y1, self.x2,
self.y2
```

```
        self.x1, self.y1 = orig_x1 - self.radius, orig_y1 - self.radius
        self.x2, self.y2 = orig_x1 + self.radius, orig_y1 + self.radius
        EllipseShape.show(self, canvas, is_preview, is_highlighted)

        self.x1, self.y1 = orig_x2 - self.radius, orig_y2 - self.radius
        self.x2, self.y2 = orig_x2 + self.radius, orig_y2 + self.radius
        EllipseShape.show(self, canvas, is_preview, is_highlighted)

        self.x1, self.y1, self.x2, self.y2 = orig_x1, orig_y1, orig_x2,
orig_y2

class CubeShape(RectShape, LineShape):

    PERSPECTIVE_RATIO = 0.4

    def __init__(self, x1, y1, x2, y2):
        Shape.__init__(self, x1, y1, x2, y2, color="blue")

    def show(self, canvas, is_preview=False, is_highlighted=False):
        orig_x1, orig_y1 = self.x1, self.y1
        orig_x2, orig_y2 = self.x2, self.y2

        RectShape.show(self, canvas, is_preview, is_highlighted)

        dx = (self.x2 - self.x1) * self.PERSPECTIVE_RATIO
        dy = (self.y2 - self.y1) * self.PERSPECTIVE_RATIO

        self.x1, self.y1 = orig_x1 + dx, orig_y1 - dy
        self.x2, self.y2 = orig_x2 + dx, orig_y2 - dy
        RectShape.show(self, canvas, is_preview, is_highlighted)

        self.x1, self.y1 = orig_x1, orig_y1
        self.x2, self.y2 = orig_x1 + dx, orig_y1 - dy
        LineShape.show(self, canvas, is_preview, is_highlighted)

        self.x1, self.y1 = orig_x2, orig_y1
        self.x2, self.y2 = orig_x2 + dx, orig_y1 - dy
        LineShape.show(self, canvas, is_preview, is_highlighted)

        self.x1, self.y1 = orig_x1, orig_y2
```

```
self.x2, self.y2 = orig_x1 + dx, orig_y2 - dy
LineShape.show(self, canvas, is_preview, is_highlighted)

self.x1, self.y1 = orig_x2, orig_y2
self.x2, self.y2 = orig_x2 + dx, orig_y2 - dy
LineShape.show(self, canvas, is_preview, is_highlighted)

self.x1, self.y1 = orig_x1, orig_y1
self.x2, self.y2 = orig_x2, orig_y2
```

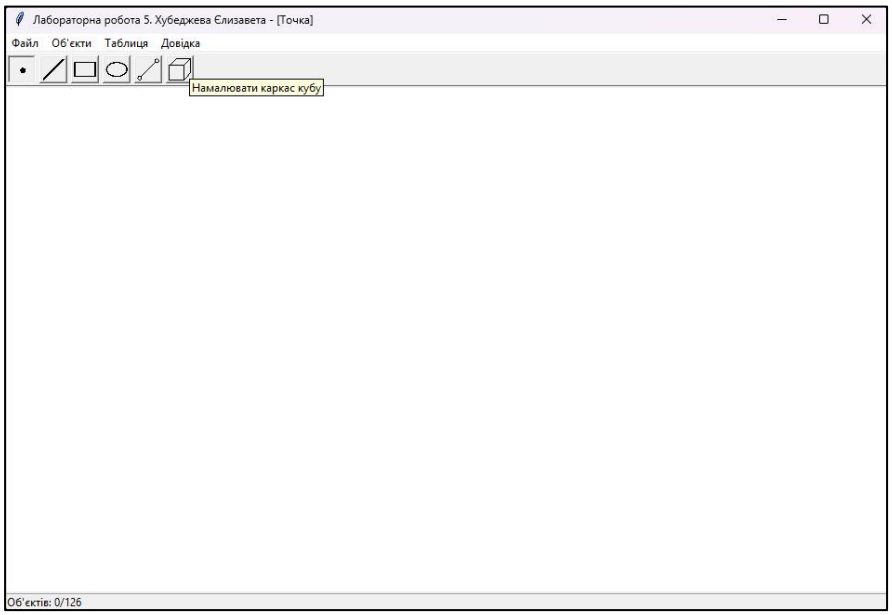
my_table.py:

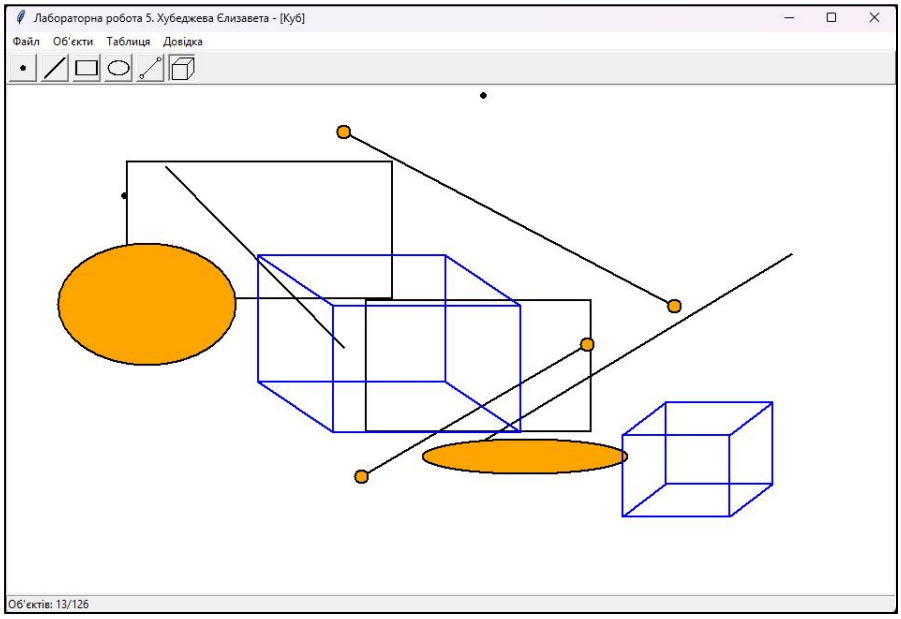
```
import tkinter as tk
from tkinter import ttk


class MyTable:


    def __init__(self, root, on_row_select_callback=None,
on_row_double_click_callback=None):
        self.window = tk.Toplevel(root)
        self.window.title("Таблиця об'єктів")
        self.window.geometry("500x300")


        self.window.protocol("WM_DELETE_WINDOW", self.hide)


        self.on_row_select_callback = on_row_select_callback
        self.on_row_double_click_callback = on_row_double_click_callback


        frame = ttk.Frame(self.window)
        frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)


        cols = ("name", "x1", "y1", "x2", "y2")
        self.tree = ttk.Treeview(frame, columns=cols, show="headings")
        self.tree.heading("name", text="Назва об'єкту")
        self.tree.heading("x1", text="X1")
        self.tree.heading("y1", text="Y1")
        self.tree.heading("x2", text="X2")
        self.tree.heading("y2", text="Y2")


        self.tree.column("name", width=120)
        for col in cols[1:]:
            self.tree.column(col, width=50, anchor=tk.E)


        scrollbar = ttk.Scrollbar(frame, orient=tk.VERTICAL,
command=self.tree.yview)
        self.tree.configure(yscrollcommand=scrollbar.set)


        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)


        self.tree.bind("<<TreeviewSelect>>", self._on_row_select)
```

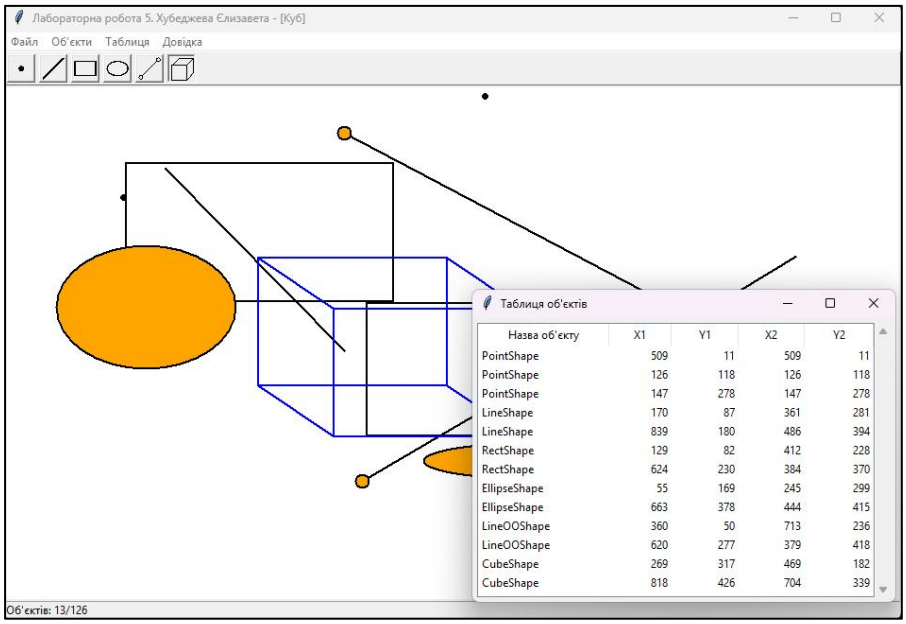
```
        self.tree.bind("<Double-1>", self._on_row_double_click)

        self.window.withdraw()

    def _get_selected_index(self):
        selected_items = self.tree.selection()
        if not selected_items:
            return None

        selected_item = selected_items[0]
        return self.tree.index(selected_item)

    def _on_row_select(self, event):
        if not self.on_row_select_callback:
            return

        index = self._get_selected_index()
        if index is not None and index >= 0:
            self.on_row_select_callback(index)

    def _on_row_double_click(self, event):
        if not self.on_row_double_click_callback:
            return

        region = self.tree.identify_region(event.x, event.y)
        if region != "cell":
            return

        index = self._get_selected_index()
        if index is not None and index >= 0:
            self.on_row_double_click_callback(index)

    def add_row(self, *columns):
        self.tree.insert("", tk.END, values=columns)
        self.tree.see(self.tree.get_children()[-1])

    def clear_table(self):
        for row in self.tree.get_children():
            self.tree.delete(row)
```

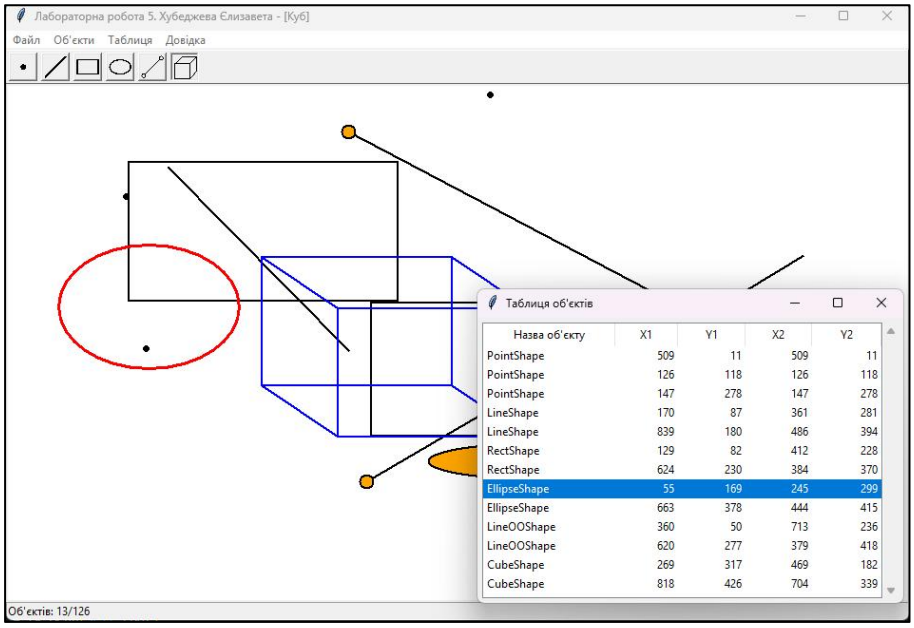
```
def show(self):
    self.window.deiconify()
    self.window.lift()

def hide(self):
    self.window.withdraw()

def destroy_window(self):
    self.window.destroy()
```

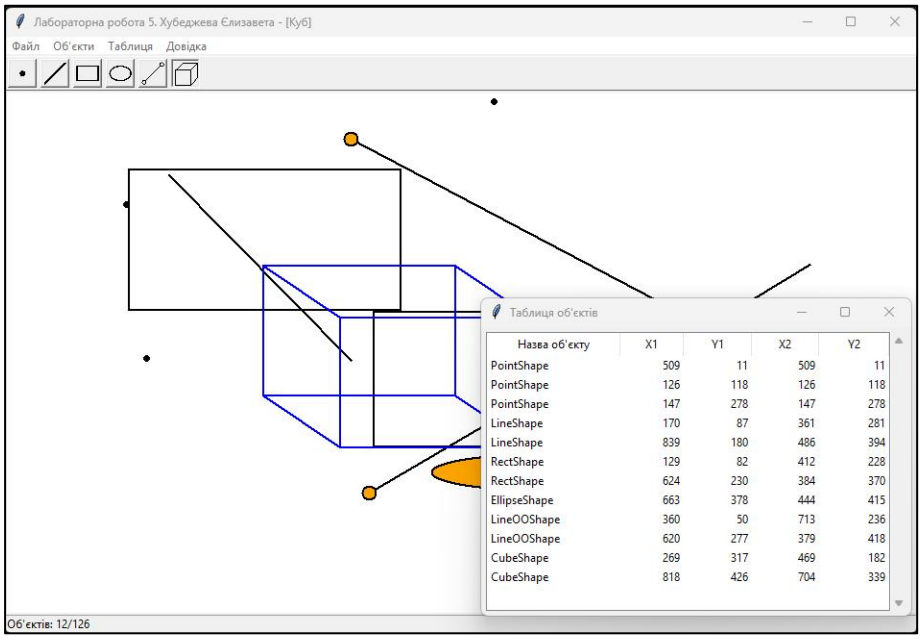
Скріншоти:


Лабораторна робота 5. Хубеджева Єлизавета - [Точка]
Файл Об'єкти Таблиця Довідка
Намалювати каркас кубу
Об'єктів: 0/126


Лабораторна робота 5. Хубеджева Єлизавета - [Куб]
Файл Об'єкти Таблиця Довідка
Об'єктів: 13/126


Лабораторна робота 5. Хубеджева Єлизавета - [Куб]
Файл Об'єкти Таблиця Довідка
Таблиця об'єктів
Назва об'єкту X1 Y1 X2 Y2
PointShape 509 11 509 11
PointShape 126 118 126 118
PointShape 147 278 147 278
LineShape 170 87 361 281
LineShape 839 180 486 394
RectShape 129 82 412 228
RectShape 624 230 384 370
EllipseShape 55 169 245 299
EllipseShape 663 378 444 415
LineOOShape 360 50 713 236
LineOOShape 620 277 379 418
CubeShape 269 317 469 182
CubeShape 818 426 704 339
Об'єктів: 13/126





shapes_log.txt:

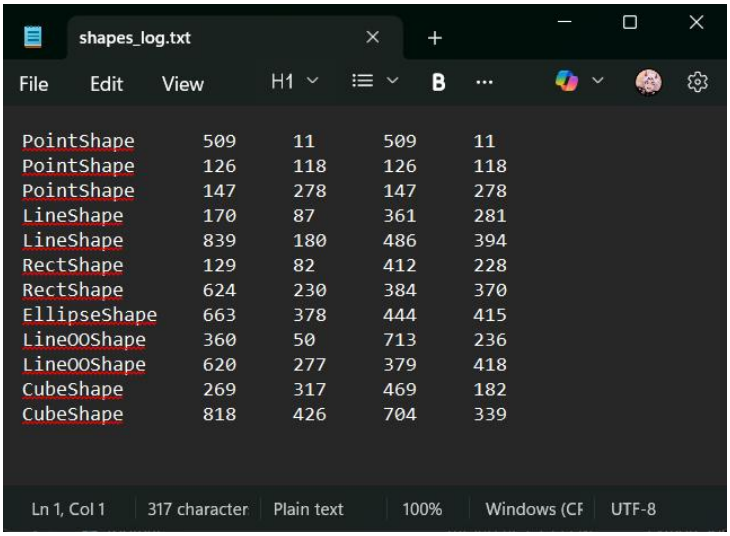
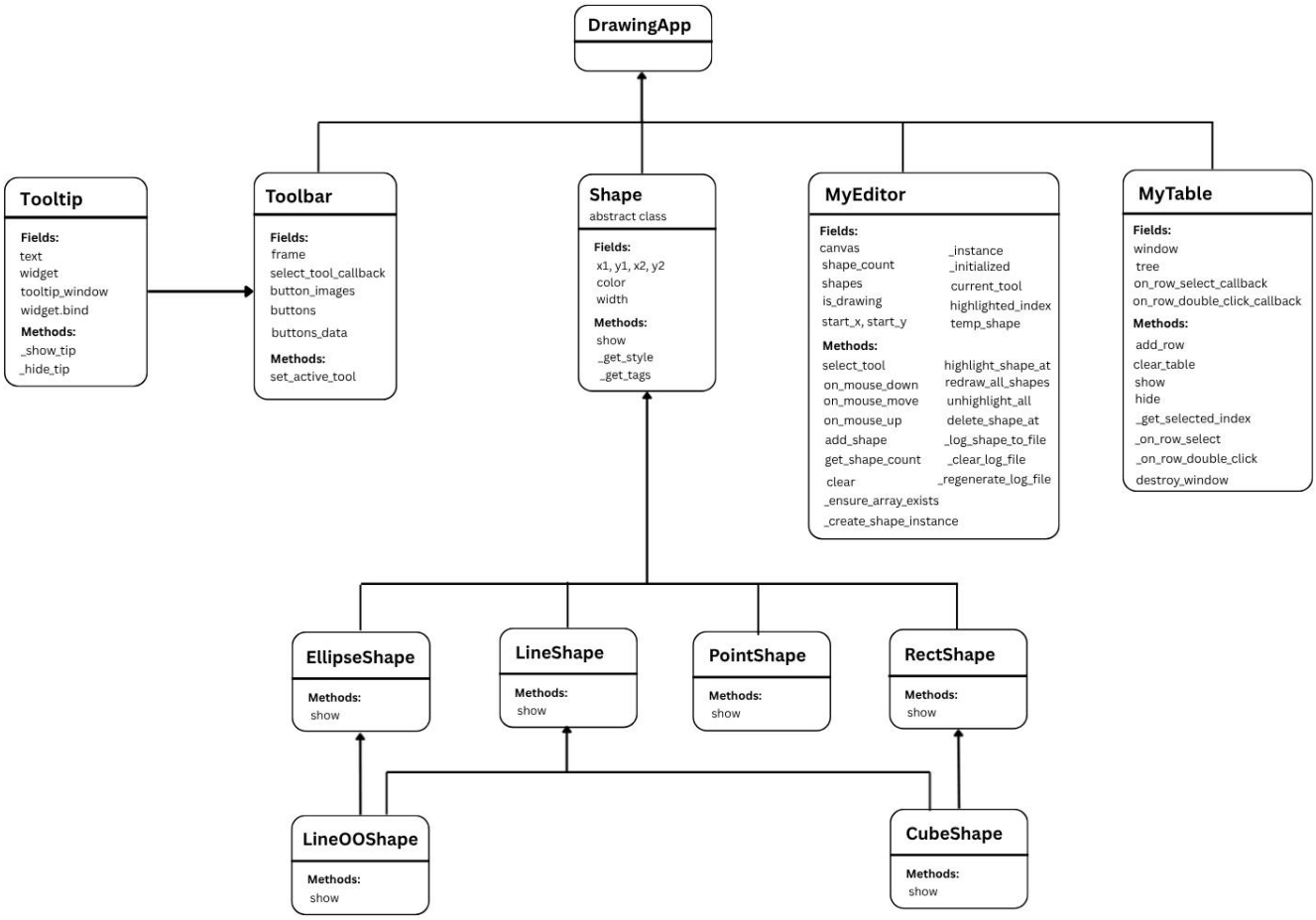


Схема успадкування класів:



Висновок

У ході виконання лабораторної роботи було продовжено розробку графічного редактора на мові Python з використанням бібліотеки Tkinter. Редактор дозволяє створювати та відображати різні геометричні об’єкти - точку, лінію, прямокутник, еліпс, а також складніші варіанти (лінія з кружечками, каркас куба). Для централізованого керування було реалізовано єдиний динамічний клас MyEditor, який інкапсулює масив об’єктів типу Shape та надає публічний інтерфейс для основних операцій графічного редактора.

Проєкт побудовано з дотриманням принципів об’єктно-орієнтованого програмування: інкапсуляції, абстракції, наслідування, поліморфізму та множинного успадкування. Абстрактний клас Shape містить поля для координат та оголошує абстрактний метод show, який конкретизують похідні класи PointShape, LineShape, RectShape, EllipseShape (а також спеціалізовані варіанти для лінії з кружечками і каркасу куба). Інтерфейсна частина реалізована в класі DrawingApp, що відповідає за меню, полотно (canvas) і статус-бар; передбачено очищення полотна та інформаційне діалогове вікно «Про програму».

Для забезпечення коректності архітектури MyEditor реалізовано за класичним шаблоном Singleton - це гарантує наявність лише одного екземпляра редактора, який централізовано керує масивом фігур і логікою їх обробки. Додатково реалізовано механізм збереження: при кожному додаванні фігури її дані (назва класу й координати) автоматично записуються у текстовий файл, що забезпечує просту форму персистентності та подальшого аналізу.

Розроблено окремий модуль my_table.py, у якому клас MyTable реалізує немодальне діалогове вікно з таблицею - у реальному часі відображається повний перелік намальованих об’єктів, їхні імена й координати з вертикальною прокруткою. Для інтерактивної взаємодії між таблицею та полотном запрограмований функціонал виділення рядка відповідної фігуру, яка змінює свій колір на червоний, а також при подвійному натисканні на вибрану фігуру можливе її видалення. Ця взаємодія реалізована через механізм callbacks, що зберігає незалежність і самодостатність модуля my_table.py.

Отже, реалізована програма поєднує чисту ООП-архітектуру і зручний інтерфейс, а модульність коду (окремі класи та незалежні модулі) забезпечує простоту підтримки й подальшого розширення функціоналу.